NITE 国家信息技术紧缺人才培养工程指定教材

教材+教案+授课资源+考试系统+题库+教学辅助案例

一站式IT就业应用系列教程

Nginx高性能Web服务器实战教程

黑马程序员 / 编著

添加QQ或微信号208695827，获取教学答案、源码，抢“助学金红包”。

本书讲解了Nginx高性能Web服务器实战技术，以及Linux、LNMP等相关内容。

提供免费教学资源，包括精美教学PPT、500道测试题、长达20小时的教学视频等。

清华大学出版社

播妞

播妞—— IT技术女神，由传智播客旗下高端教育品牌黑马程序员推出，专门服务于计算机相关专业的大学生及IT爱好者；可随时提供教材源代码、习题答案、免费视频教程和就业宝典等。

播妞倾情寄语：

你加，或者不加我
我就在这里
不离 不弃
来我这里
或者
让我住进你的心里

★ 小心！此处常有“助学金红包”出没！★
快来领取！

播妞QQ：208695827

播妞微信：208695827

教师获取教材配套资源

添加微信/QQ

2011168841

Nginx高性能Web服务器实战教程

黑马程序员 / 编著

清华大学出版社
北 京

内 容 简 介

Nginx是目前备受关注的一个高性能 HTTP 和反向代理服务器，具有简单、灵活的配置和极高的执行效率。本书面向 Linux 运维方向的用户或具备某一类编程语言(C、PHP、Java)基础的读者，旨在使其快速掌握 Nginx 的配置与应用，学会搭建高性能的 Web 服务器。

全书分为 8 章，主要内容包括 VMware 虚拟机的使用、Linux 入门、正则表达式、HTTP 协议、Nginx 安装与配置、访问控制、日志管理、虚拟主机、Web 服务器搭建、反向代理、负载均衡、缓存以及一些常用模块和应用(包括调试输出、网页压缩、重写、重定向、防盗链、HTTPS 等)。第 8 章还着重介绍了 Nginx 的配置优化、LNMP 分布式集群和高可用方案的部署，目的是帮助读者将 Nginx 应用到复杂的服务器架构中，对所学知识进行巩固和提高。

本书是一本 Nginx 的入门书籍，适合作为高等院校本、专科计算机相关专业的教材，也可作为 Nginx 爱好者的参考书。

图书在版编目(CIP)数据

Nginx 高性能 Web 服务器实战教程/黑马程序员编著. —北京：清华大学出版社，2017
ISBN 978-7-302-47244-5

Ⅰ. ①N…　Ⅱ. ①黑…　Ⅲ. ①互联网络－网络服务器－教材　Ⅳ. ①TP368.5

中国版本图书馆 CIP 数据核字(2017)第 126528 号

责任编辑：袁勤勇　徐跃进
封面设计：马丹
责任校对：焦丽丽
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市吉祥印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：16.75　　**字　　数**：405 千字
版　　次：2017 年 9 月第 1 版　　**印　　次**：2017 年 9 月第1 次印刷
印　　数：1～2000
定　　价：45.00元

产品编号：075059-01

序言

传智播客和“黑马程序员”

江苏传智播客教育科技股份有限公司(简称传智播客)是一家专门致力于高素质软件开发人才的科技公司,“黑马程序员”是传智播客旗下高端 IT 教育品牌。

“黑马程序员”的学员多为大学毕业后想从事 IT 行业,但各方面条件还不成熟的年轻人。“黑马程序员”的学员筛选制度非常严格,包括严格的技术测试、自学能力测试,还包括性格测试、压力测试、品德测试等。百里挑一的残酷筛选制度确保了学员质量,并降低了企业的用人风险。

自“黑马程序员”成立以来,教学研发团队一直致力于打造精品课程,不断在产、学、研三个层面创新自己的执教理念与教学方针,并集中“黑马程序员”的优势力量,有针对性地出版计算机教材五十多种,制作教学视频数十套,发表各类技术文章数百篇。

“黑马程序员”不仅研发 IT 系列教材,还为高校师生提供以下配套学习资源与服务。

为大学生提供的配套服务:

- 专注的辅学平台“博学谷”(http://yx.boxuegu.com),专业老师在线为您答疑解惑。
- 针对高校学生在学习过程中存在的压力等问题,我们还为大学生量身打造了“播妞”。“播妞”不仅致力推行快乐学习,还有定期的助学红包雨(“播妞”微信/QQ:208695827)。
- 高校学生也可扫描右方二维码,加入“播妞”粉丝团,获取最新学习资源,与“播妞”一起快乐学习。

为 IT 教师提供的配套服务:

针对高校教学,“黑马程序员”为 IT 系列教材精心设计了“教案+授课资源+考试系统+题库+教学辅助案例”的系列教学资源,高校老师请关注码大牛老师微信/QQ:2011168841,获取教材配套资源,也可以扫描右方二维码,加入专为 IT 教师打造的师资服务平台——“教学好助手”,获取“黑马程序员”最新教师教学辅助资源及相关动态。

前言

Nginx 自从发布以来，在世界范围内受到越来越多的关注。由于其稳定性好、功能丰富、占用内存少、并发能力强等优势，在同类型的服务器中表现出色。一些大型网站如百度、京东、新浪、网易、腾讯、淘宝等都已经在内部广泛使用。因此，对于这款高性能、轻量级的 HTTP 和反向代理服务器，如何能够快速、系统地了解并掌握它的应用，成为初学者的迫切需求。

为什么要学习本书

本书针对的是以 Linux 运维为方向或具备某一类开发语言（如 C、PHP、Java 等）基础的读者。全书围绕 Nginx 功能使用以及重点配置案例展开，并铺垫了 Linux 服务器、正则表达式、HTTP 等方面的基础知识，适合想要快速掌握 Nginx 的初学者。

教材遵循学习的难易度及先后顺序来安排章节顺序，按照“概念讲解＋案例演示”的方法来设计课程内容，将抽象的概念具体化，学到的知识实践化，让读者不仅理解知识内容，还能根据实际需求进行分析和处理，达到学以致用、学用结合的效果。

如何使用本书

本书的主要内容包括 VMware 虚拟机、Linux 系统、正则表达式、HTTP 协议、Nginx 安装与配置、Web 服务器搭建、反向代理、负载均衡、缓存，以及 Nginx 常用模块的使用、Nginx 配置优化、LNMP 分布式集群、Nginx＋Keepalived 高可用方案等。

全书共分为 8 章：

- 第 1 章主要讲解常见的 Web 服务器、Linux 系统以及 VMware 虚拟机的使用，通过本章学习，读者可以了解一些基本概念，并能够搭建一个实验环境。
- 第 2 章讲解 Linux 入门、正则表达式和 HTTP 协议，这些是学习 Nginx 必备的基础知识，只有掌握这些内容，才能够在 Nginx 使用过程中得心应手。
- 第 3 章讲解 Nginx 的安装和服务器环境的配置，主要包括网络和防火墙配置、远程终端的使用、服务脚本的编写、软件的安装等。通过本章的学习，读者能够掌握在最小方式安装的 CentOS 系统中编译安装 Nginx，并对服务器进行管理和维护。
- 第 4 章讲解 Nginx 服务器的基本配置，认识 Nginx 配置文件并了解一些基本指令的作用。通过具体案例实现了访问控制、日志管理和各种类型虚拟主机。
- 第 5 章介绍 Web 服务器的搭建，包括 Nginx 与 PHP、Apache、Tomcat 等软件的组

合搭配，以及基于 Nginx＋Lua 的高性能 Web 开发平台 OpenResty 环境的使用。

- 第 6 章讲解反向代理、负载均衡、缓存配置以及邮件服务，为读者展示了如何解决大型网站多台服务器之间协同工作的问题，以便提高计算机系统的处理能力、计算速度，从而满足业务量的需求。
- 第 7 章从模块的角度介绍 Nginx 相关应用，包括调试输出、查看响应状态、替换响应内容、网页压缩传输、重写、重定向、防盗链、HTTPS 等配置。通过本章的学习，读者能够掌握根据具体需求来对 Nginx 进行不同的模块编译和配置。
- 第 8 章主要讲解 Nginx 配置优化、LNMP 分布式集群以及 Nginx＋Keepalived 高可用方案的部署。本章是对 Nginx 的综合应用和提升，从整体服务器架构上满足大型网站对高并发、高负载和高可用的需求。

在上面所列举的 8 章中，第 1～3 章是基础课程，主要帮助初学者掌握扎实的基本功；第 4、6、7 章是 Nginx 的重点课程，讲解 Nginx 的基本配置、常用模块和典型应用；第 5、8 章是动手实战课程，涉及 Apache、Tomcat、OpenResty、PHP、MySQL、Memcached、NFS、Keepalived 等多种软件和服务的搭建与配置，这部分内容比较复杂，希望读者细心阅读，灵活运用 VMware 虚拟机的快照和克隆功能，认真完成每个操作步骤。

在学习过程中，读者一定要亲自实践教材中演示的案例。如果不能完全理解书中所讲知识，读者可以登录博学谷平台，通过平台中的教学视频进行深入学习。读者在学习完一个知识点后，要及时在博学谷平台上进行测试，以巩固学习内容。

另外，如果读者在理解知识点的过程中遇到困难，建议不要纠结于某个地方，可以先往后学习。通常来讲，通过逐渐的学习，对于前面不懂和疑惑的知识也能够理解。在学习 Nginx 的过程中，一定要多多动手实践，如果在实践的过程中遇到问题，建议多思考，理清思路，认真分析问题发生的原因，并在问题解决后及时总结经验。

致谢

本书的编写和整理工作由传智播客教育科技股份有限公司的黑马程序员完成，主要参与人员有吕春林、韩冬、乔治铭、高美云、陈欢、马丹、王哲、李东超、韩振国、王金涛等，全体参编人员在这近一年的编写过程中付出了很多辛勤的劳动，特此表示衷心的感谢。

意见反馈

尽管我们付出了最大的努力，但教材中难免会有不妥之处，欢迎各界专家和读者朋友们来信、来函给予宝贵意见，我们将不胜感激。您在阅读本书时，如发现任何问题或有不认同之处可以通过电子邮件与我们取得联系。请发送电子邮件至：itcast_book@vip.sina.com。

黑马程序员

2017 年 4 月

目录

第1章 Nginx开篇

学习目标

- 熟悉 Nginx 的特点和主要应用；
- 熟悉 Linux 操作系统的安装和使用；
- 掌握 VMware 虚拟机的使用。

在开始学习 Nginx 之前，首先要对 Web 服务器有一定了解，其次要明确 Nginx 与其他 Web 服务器的区别，最后要掌握 Nginx 的主要运行平台 Linux 的安装与操作。本章将对 Nginx 的主要特点和应用进行概述，并对如何利用 VMware 虚拟机模拟 Linux 环境，利用快照功能实现系统的备份还原等操作进行详细讲解。

1.1 Nginx 简介

1.1.1 Web 服务器

Web 服务器又称为 WWW(World Wide Web，万维网)服务器，或网站服务器，主要用于提供网上信息浏览服务，Nginx 就是一款高性能的 Web 服务器软件。

目前，Linux 和 Windows Server 操作系统是搭建 Web 服务器最为常见的系统。其中，Linux 系统的特点是安全性高且代码开源，是较为理想的 Web 服务器操作系统，在该平台下常见的 Web 服务器软件有 Apache、Tomcat、Lighttpd 和 Nginx 等；而在 Windows Server 平台下常见的 Web 服务器软件是 Microsoft IIS。

1. Apache

Apache(Apache HTTP Server)是目前广泛流行的 Web 服务器软件，具有开放源代码、跨平台、安全稳定等特点。Apache 是伴随互联网的兴起共同成长的，经过多年的技术沉淀和积累，已经非常成熟和稳定，具备了大量的功能模块和扩展。但由于 Apache 在设计之初对性能和资源的消耗没有过多的关注，导致在应对高并发的业务场景时，被一些轻量级的高性能 Web 服务器赶超。

2. Tomcat

Tomcat(Apache Tomcat)主要用于 Java Web 环境，是一个运行 Servlet 和 JSP 的容器(即运行 Java 语言的服务器端程序)。Tomcat 和 Apache 都是由 Apache 软件基金会运作的开源项目，Tomcat 本身可作为一个单独的 Web 服务器使用，主要用于处理动态请求，但在

静态资源和高并发方面的性能较弱，因此经常和 Apache 等软件搭配，实现动静态请求分离。

3. Lighttpd

相对于 Apache 服务器，由德国人发起的轻量级开源 Web 服务器软件 Lighttpd，不仅实现了 Apache 的常用功能，同时还保持了轻量级的优势，具有低内存开销、低 CPU 占用率、性能高以及模块丰富等特点。目标是专门针对高性能网站提供一个安全、快速、兼容性强且配置灵活的 Web 服务器环境。

4. Nginx

Nginx(读作 engine x)是一个轻量级开源 Web 服务器软件，可以作为反向代理、负载均衡与缓存服务器使用。Nginx 和 Lighttpd 都是为高并发网站的应用场景而设计的。随着技术发展和业务需要，Nginx 逐渐受到关注，在国内如百度、淘宝、腾讯、新浪、网易等网站都开始使用 Nginx 来满足一些高并发访问的需求。

5. Microsoft IIS

IIS(Internet Information Services，互联网信息服务)是 Microsoft(微软)公司的 Web 服务器产品，运行在 Windows Server 平台，具有图形界面管理工具。IIS 是目前被广泛采用的 Web 服务器软件之一，在全球占有相当大的市场份额。IIS 和 Windows Server 组合可以提供可靠、完整的网络服务器解决方案，但作为付费软件，需要支付一定的软件成本。

1.1.2 Nginx 概述

Nginx 是俄罗斯人 Igor Sysoev 开发的一个开源的高性能 Web 服务器软件，起初是为 Rambler.ru(俄罗斯访问量第二的大型门户网站和搜索引擎)使用的，具有轻量级和高并发的特点，第一个公开版本 0.1.0 发布于 2004 年 10 月。目前，Nginx 通过 BSD-like 开源软件许可协议发行，可以在 UNIX、Linux、macOS、Solaris 以及 Windows 等操作系统中运行。

根据 2016 年 6 月份 Netcraft 公司统计的数据显示，在全球约 10 亿网站中，市场占有率排名前三的 Web 服务器软件分别是 Apache(34%)、Microsoft IIS(32%)和 Nginx(16%)，可以看到作为后起之秀的 Nginx 增长速度非常快。Nginx 之所以能够脱颖而出，是因为它具有性能高、稳定性好、结构模块化、配置简单以及资源消耗非常低的优点。

Nginx 可以提供 HTTP 服务，包括处理静态文件，支持 SSL(提供 HTTPS 访问)、GZIP(网页压缩)、虚拟主机、URL 重写等功能，可以搭配 FastCGI 程序(如 PHP)处理动态请求。除此之外，Nginx 还可以用于代理、反向代理、负载均衡、缓存等服务器功能，在集群环境中解决网络负载、提高可用性等。关于这些功能的使用在本书的后面有详细的讲解。

Nginx 的官方网站是 https://nginx.org，通过浏览器打开后如图 1-1 所示。

从图 1-1 中可以看到 Nginx 的版本更新情况。网页右方的侧边栏是导航链接，通过 about 链接可以查看 Nginx 的基本信息，download 链接可以获取 Nginx 的下载地址，documentation 链接可以阅读 Nginx 的参考文档。Nginx 的文档是非常详细的参考手册，用于查阅各模块和指令的详细解释。

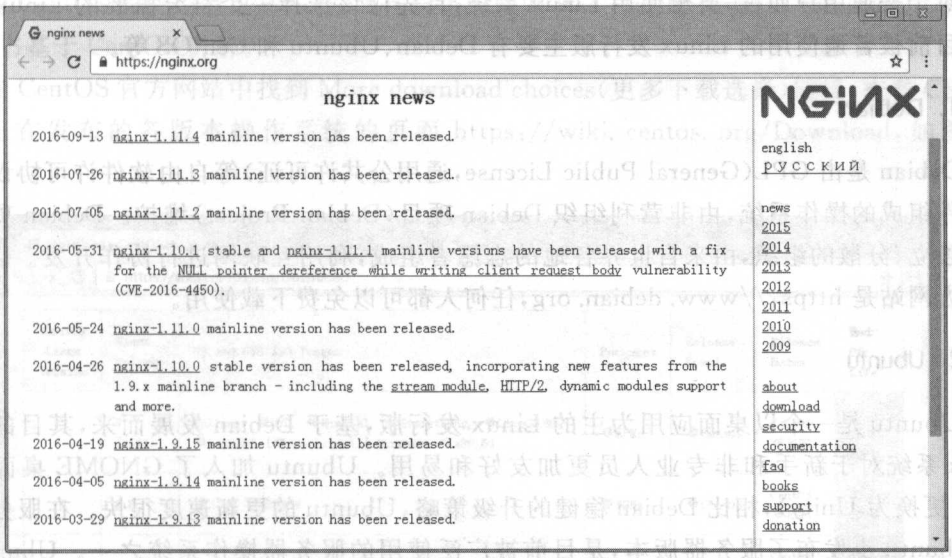


图 1-1 Nginx 网站首页

1.2 Linux 操作系统

Linux 是目前最适合运行 Nginx 的操作系统，尤其是 Linux 内核加入了 epoll(一种事件模型)机制以后，服务器支持的并发量大幅提升，Nginx 能够高效地支持上万级的高并发连接正是得益于此。本节将针对 Linux 操作系统进行详细介绍。

1.2.1 Linux 的起源与发展

Linux 是一种开放源代码和自由传播的计算机操作系统，其目的是建立不受任何商品化软件版权制约、全世界都能自由使用的 UNIX 兼容产品。严格来说，Linux 这个词本身只表示 Linux 内核，但是人们已经习惯使用 Linux 来形容整个基于 Linux 内核，并且使用 GNU 计划中众多外围程序的操作系统。

Linux 内核由 Linus Torvalds(林纳斯·托瓦兹)在 1991 年 10 月 5 日首次发布，最初是作为 Intel x86 架构个人计算机的一个自由操作系统，后来被移植到更多的计算机硬件平台，在服务器、超级计算机、嵌入式系统等领域都有广泛应用。在互联网和智能设备高速发展的今天，围绕人们生活的手机、平板电脑、路由器、电视机等智能设备都可能搭载了 Linux 系统。例如，在移动设备上广泛使用的 Android 操作系统就是建立在 Linux 内核之上。目前，Linux 内核由 https://kernel.org 网站对其进行维护。

Linux 系统是开源和自由的，因此发展出了各种各样的版本，同时也在遵循一定的规范。Linux 有许多发行版，即由一些团体、公司或个人为了不同目的而制作的版本，通常由 Linux 内核和许多外围软件组成。在规范上，Linux 属于类 UNIX 系统(与传统 UNIX 操作系统相似)，各种版本在一定程度上都遵守 POSIX(Portable Operating System Interface，可移植操作系统)规范。

对于普通用户而言，要想使用 Linux 系统，首先应该选择一个符合需要的 Linux 发行版。目前被普遍使用的 Linux 发行版主要有 Debian、Ubuntu 和 CentOS 等。

1. Debian

Debian 是由 GPL(General Public License，通用公共许可证)等自由软件许可协议授权的软件组成的操作系统，由非营利组织 Debian 项目(Debian Project)维护。Debian 项目是一个独立、分散的组织，由来自世界各地的志愿者组成，利用互联网进行协作开发。Debian 的官方网站是 https://www.debian.org，任何人都可以免费下载使用。

2. Ubuntu

Ubuntu 是一个以桌面应用为主的 Linux 发行版，基于 Debian 发展而来，其目的是让 Linux 系统对于新手和非专业人员更加友好和易用。Ubuntu 加入了 GNOME 桌面环境(后来更换为 Unity)，相比 Debian 稳健的升级策略，Ubuntu 的更新速度很快。在服务器领域，Ubuntu 也发布了服务器版本，是目前被广泛使用的服务器操作系统之一。Ubuntu 的中文官方网站是 http://cn.ubuntu.com，可以免费下载使用。

3. Red Hat Enterprise Linux

Red Hat Enterprise Linux 是 Red Hat 公司开发的一款面向商业市场的 Linux 发行版，属于商业软件。与免费下载使用的 Linux 系统不同的是，购买 Red Hat Enterprise Linux 操作系统可以获得 Red Hat 公司的商业性的技术支持，对于需要付费服务的企业而言，可以考虑选择这款操作系统。

4. Fedora

Fedora 是知名度较高的 Linux 发行版之一，由 Fedora 项目社区开发，Red Hat 公司提供赞助。Fedora 基于 Red Hat Linux 操作系统发展而来，在 Red Hat Linux 终止发行后用来替代其在个人领域的应用，并另外发行 Red Hat Enterprise Linux 用于商业领域。对于普通用户而言，Fedora 是一套功能完备、更新快速的免费操作系统，对于 Red Hat 公司而言，它是许多新技术的测试平台，被认可的技术会加入到商业系统中。在 Fedora 官方网站 https://getfedora.org 可以获取系统的下载地址。

5. CentOS

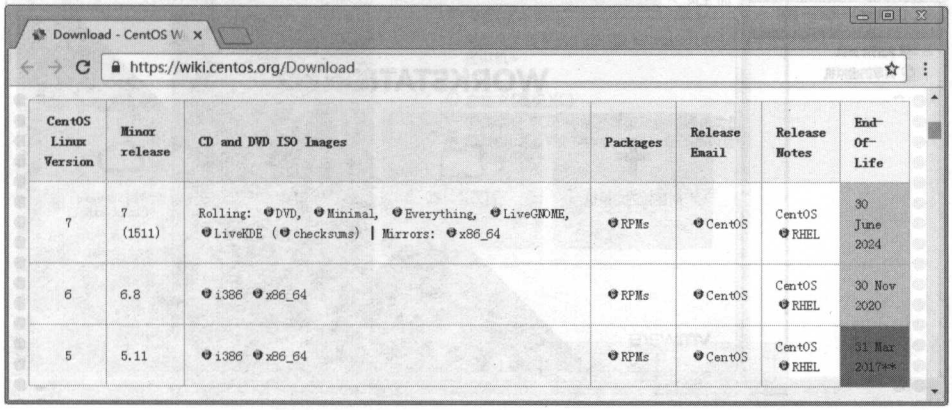
CentOS(Community Enterprise Operating System)是来自于 Red Hat Enterprise Linux 依照开放源代码规定所发布的源代码编译成的系统，因此两个系统都出自相同的源代码，不同之处在于 CentOS 不包含封闭源代码的软件，且没有 Red Hat 的商业技术支持。目前 CentOS 由 CentOS 项目(CentOS Project)组织负责维护，官方网站为 https://www.centos.org，可以免费下载使用。

1.2.2 获取 Linux 系统

在 Linux 各种发行版中，Ubuntu 和 CentOS 都是免费且非常优秀的服务器操作系统，

在国内 CentOS 的使用量更高一些，而且与企业版的 Red Hat Enterprise Linux 使用非常相似，本书基于 CentOS 讲解。

在 CentOS 官方网站中找到 More download choices(更多下载选择)链接，查看 CentOS 目前正在发布的各版本操作系统的页面 https://wiki.centos.org/Download，如图 1-2 所示。



CentOS Linux Version	Minor release	CD and DVD ISO Images	Packages	Release Email	Release Notes	End-Of-Life
7	7 (1511)	Rolling: DVD, Minimal, Everything, LiveGNOME, LiveKDE (checksums) | Mirrors: x86_64	RPMs	CentOS	CentOS RHEL	30 June 2024
6	6.8	i386 x86_64	RPMs	CentOS	CentOS RHEL	30 Nov 2020
5	5.11	i386 x86_64	RPMs	CentOS	CentOS RHEL	31 Mar 2017**

图 1-2　获取 CentOS

从图 1-2 中可以看出，目前 CentOS 正在发布的版本共分为 3 个系列，其中 7 系列最新版本为 7(1511)，6 系列最新版本为 6.8，5 系列最新版本为 5.11。表格最后一列的 End-Of-Life 表示该版本停止维护的时间，通常各系列版本的维护周期为 7～10 年。

CentOS 的 5 系列和 6 系列又分为 i386 和 x86_64 两种类型，分别对应处理器的 32 位和 64 位(64 位兼容 32 位)，而 7 系列只提供了 x86_64 版本。本书基于 CentOS 6.8 x86_64 版本进行讲解，在网站中找到 CentOS-6.8-x86_64-bin-DVD1.iso 系统光盘镜像文件进行下载即可。

1.3　VMware 虚拟机

在获取 CentOS 操作系统镜像以后，最直接的方法就是将镜像刻录成光盘，然后通过光盘为计算机安装操作系统。从学习的角度来看，这种方式不仅麻烦而且成本也高。此时，可以借助一些虚拟机软件，在一台计算机中虚拟出多台计算机，随意安装各种操作系统，而不必额外购置任何硬件，省时省力。本节将针对 VMware 虚拟机进行讲解。

1.3.1　什么是虚拟机

虚拟机(Virtual Machine)是指通过软件模拟出的具有完整硬件系统功能、运行在一个完全隔离环境中的完整计算机系统。通常人们身边只有一台供自己使用的计算机，而通过虚拟机软件可以在一台计算机中虚拟出多台计算机，每台虚拟的计算机都可以定制不同的硬件环境，安装不同的操作系统。虚拟机的性能取决于物理机的性能，并且虚拟化技术本身会带来一定的性能下降，因此对物理机的硬件要求比较高。

VMware Workstation(简称 VMware)是一款非常优秀的虚拟机软件，对于开发人员和

系统管理人员而言，VMware 在虚拟网络、实时快照、共享文件夹等方面的特点使其成为一个开发、测试、部署环境和应用程序的最佳解决方案。VMware 可以运行于 Windows 环境，图 1-3 演示了 VMware 的软件界面。

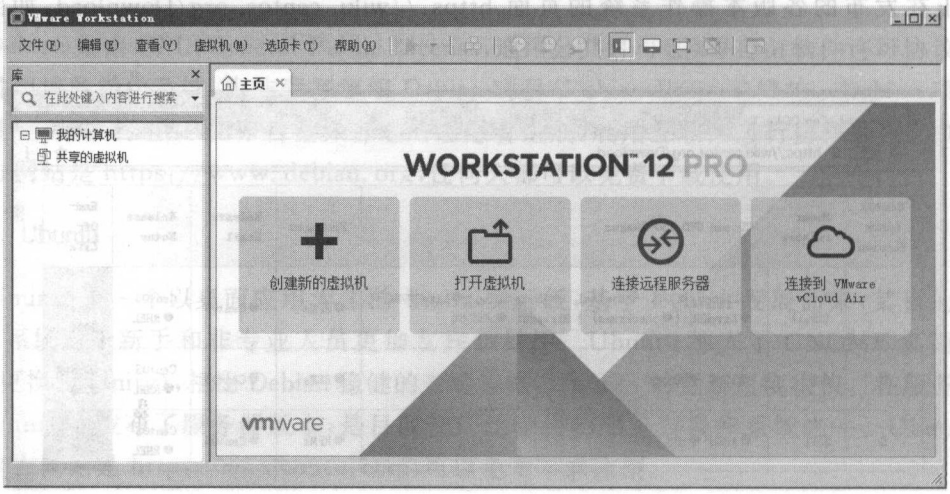


图 1-3 VMware 软件界面

1.3.2 VMware 安装 CentOS

在准备好 VMware 虚拟机软件之后，接下来开始安装 CentOS 系统，具体操作步骤如下。

(1) 在安装系统之前，需要创建一台虚拟机，并配置硬件参数。在 VMware 中执行菜单栏“文件”→“新建虚拟机”命令，如图 1-4 所示。

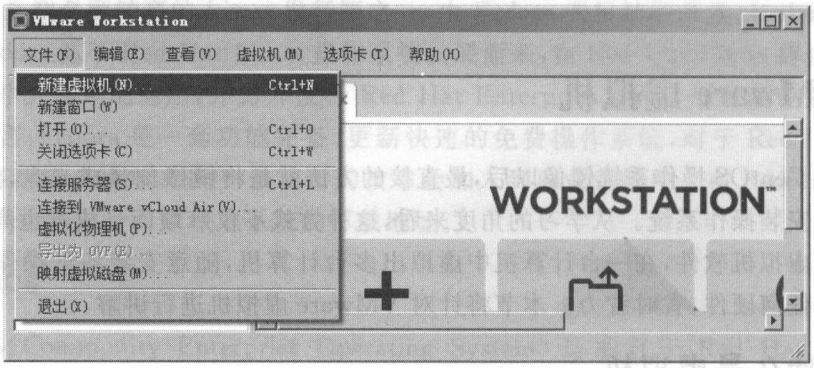


图 1-4 新建虚拟机

(2) 在弹出的“新建虚拟机向导”页面中选择“典型(推荐)”，然后单击“下一步”按钮，如图 1-5 所示。

(3) 在“安装客户机操作系统”页面，选择“安装程序光盘映像文件”单选按钮，然后浏览找到 CentOS 的光盘镜像文件 CentOS-6.8-x86_64-bin-DVD1.iso，如图 1-6 所示。完成后单击“下一步”按钮。

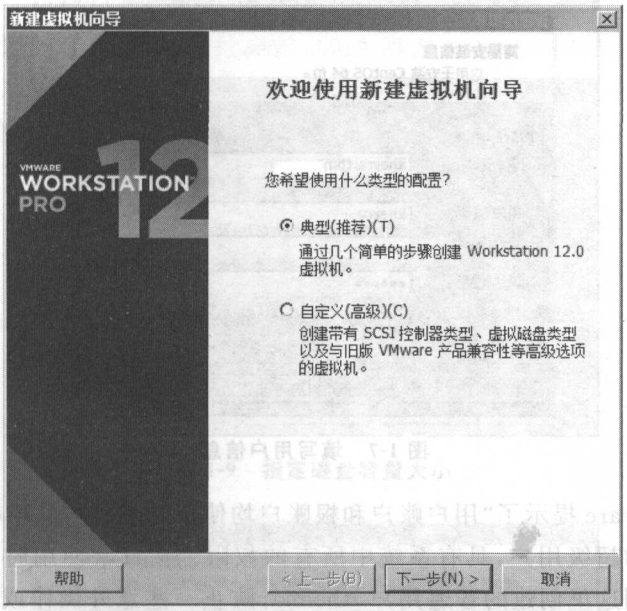


图 1-5 “新建虚拟机向导”对话框

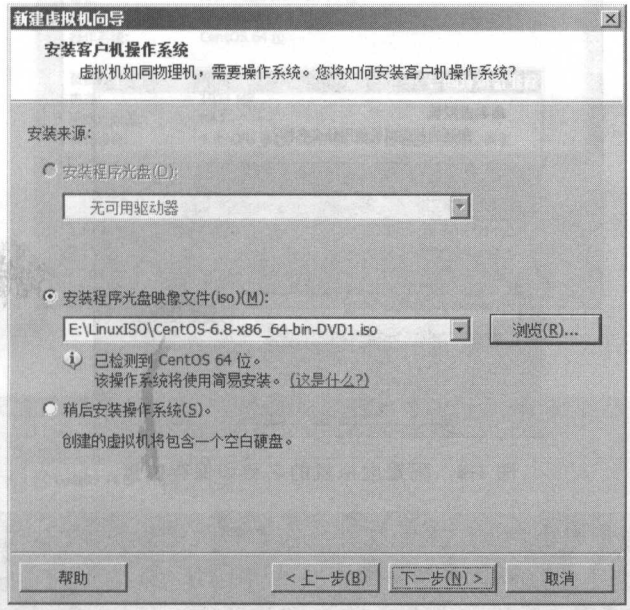


图 1-6 选择系统镜像

在图 1-6 中,VMware 提示“已检测到 CentOS 64 位。该操作系统将使用简易安装。”简易安装是 VMware 提供的一个自动安装操作系统的功能,当虚拟机创建完成并开机后会自动安装系统,无须手动操作,并同时自动为 CentOS 系统安装 VMware 工具 VMware Tools 实现一些虚拟机的增强功能。

(4) 在“简易安装信息”页面,填写 CentOS 系统安装后自动创建的用户信息,如图 1-7 所示。其中“用户名”用于登录系统和内部程序识别,“全名”仅用于友好显示。

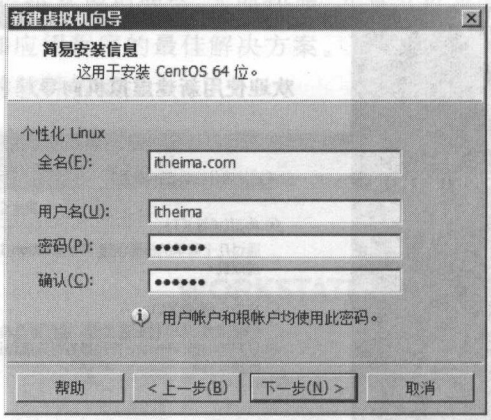


图 1-7 填写用户信息

图 1-7 中 VMware 提示了“用户账户和根账户均使用此密码”，其中根账户是指 root 用户，是系统中唯一的超级用户，具有系统中所有的权限。在日常使用时一般不会直接使用 root 用户，而是使用这里创建的普通用户 itheima，防止一些意外操作或恶意软件破坏系统安全。

(5) 配置虚拟机的名称和保存位置，应选择一个较大的硬盘分区来保存虚拟机文件，如图 1-8 所示。

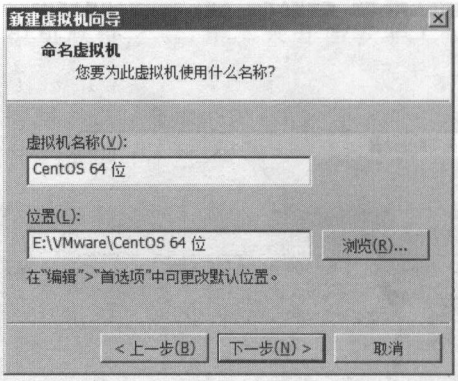


图 1-8 配置虚拟机的名称和保存位置

(6) 指定磁盘容量大小。由于本课程并不需要很大的磁盘空间，因此配置 10GB 的空间就足够了。如果物理机的磁盘文件系统支持大于 4GB 以上的单文件(如 NTFS 文件系统)，则选择“将虚拟磁盘存储为单个文件”单选按钮性能更好，如图 1-9 所示。

(7) 在“已准备好创建虚拟机”页面可以进行硬件定制，如图 1-10 所示。

此处读者可以根据物理机的硬件能力定制虚拟机的硬件，推荐设置虚拟机的 CPU 核心数量与物理机一致，从而使虚拟机的性能更好；内存至少 1GB，其他保持默认值即可。

(8) 完成上述配置后，单击“完成”按钮开始创建虚拟机，创建后会自动开启并自动安装 CentOS，安装过程如图 1-11 所示。

在图 1-11 中，“CentOS 64 位”标签页的下方是虚拟机开机后显示的画面，当前 CentOS 系统正在进行安装。单击虚拟机的画面即可将键盘和鼠标输入定向到虚拟机内，当需要恢

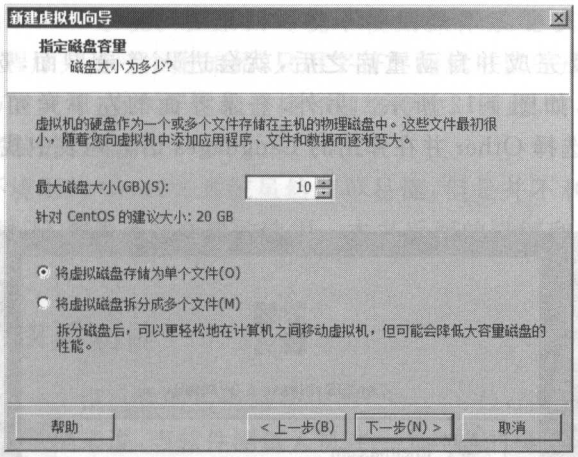


图 1-9 指定磁盘容量大小

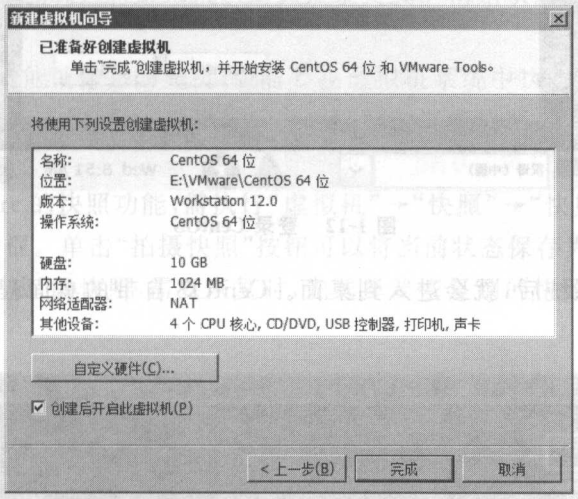


图 1-10 “已准备好创建虚拟机”页面

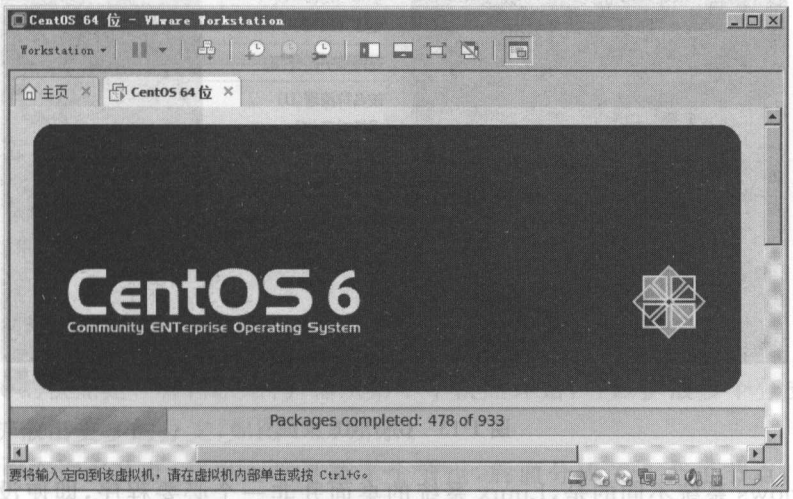


图 1-11 正在安装 CentOS

复时，按 Ctrl＋Alt 键即可。

(9) CentOS 安装完成并自动重启之后，就会进入登录界面，选择之前创建的用户 itheima 并输入密码，如图 1-12 所示。另外，登录界面的左下角可以切换语言，默认是 English(英语)，可以选择 Other 并在弹出的 Languages 语言列表中找到“汉语(中国)”进行切换。

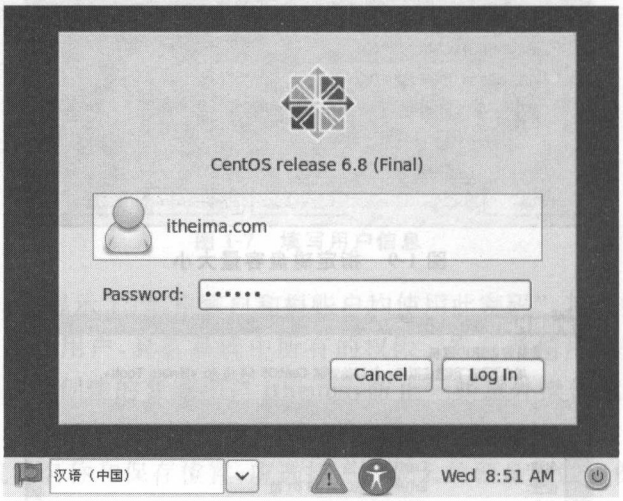


图 1-12 登录 CentOS

(10) 成功登录系统后，就会进入到桌面。CentOS 自带的桌面程序为 GNOME 2.28，如图 1-13 所示。

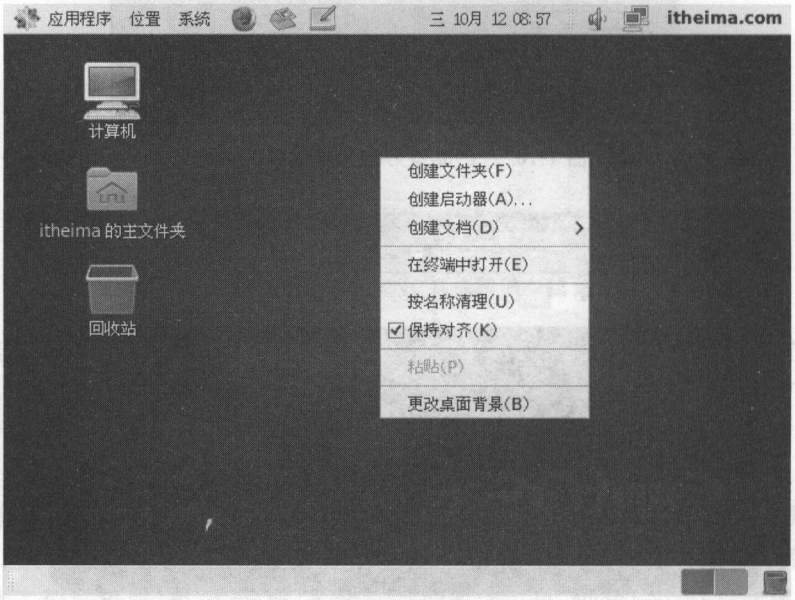


图 1-13 GNOME 桌面

和 Windows 系统不同的是，Linux 系统的桌面并非一个必要程序，即使没有桌面依然

可以用字符界面控制 Linux 系统。在桌面环境下使用操作系统非常简单方便，但对于 Linux 系统而言，桌面程序只是一个附加品，只用字符界面就可以完成所有的操作，而且比图形界面更稳定、更节省资源，有利于远程连接和网络传输。因此，许多 Linux 服务器不安装桌面，只通过远程终端进行操作。

另外，中文语言环境下的 CentOS 系统虽然简单易懂，但是并不利于学习，这里推荐读者切换到英文语言环境下，熟悉英文环境的使用。本书后面的讲解和案例都是基于英文语言环境下进行的。

1.3.3　VMware 快照功能

VMware 虚拟机的一个很重要的功能就是快照(Snapshot)，简而言之就是一种快速的系统备份与还原功能。举例来说，当软件测试人员在一个纯净的操作系统中测试时，一旦软件出现问题破坏了系统文件，就需要重新安装系统。而利用快照功能，可以在安装系统后创建一个快照，每次版本更新时，将虚拟机回退到系统安装的初始状态，从而彻底避免测试环境出现干扰因素。

VMware 的快照功能就像是时光机，当需要在虚拟机系统中执行某个难以预料的操作时，可以在执行前创建一个快照，一旦执行后出现问题就恢复快照，从而极大地提高了测试效率。

若要使用 VMware 的快照功能，需执行“虚拟机”→“快照”→“快照管理器”操作，打开如图 1-14 所示的对话框。单击“拍摄快照”按钮可以将当前状态保存为快照，图中已经创建了一个名称为“安装完成”的快照，需要恢复时，先选择要恢复到的快照，再单击“转到”按钮即可。

图 1-14　快照管理器

VMware 的快照是一种树状结构，即从第 1 个快照开始，可以扩散多个分支快照，每个分支又可以扩散出更多的分支，如图 1-15 所示。

图 1-15 中有许多节点，其中入学是第 1 个节点，后面分成了 3 个子节点，分别是学习 Linux、学习 Java、学习 PHP，选择不同的节点决定了后面不同的发展。这些节点就是虚拟

机创建的快照，每个快照都可以恢复成当前位置，也可以基于某个快照创建更多的子级快照。假如有一个新学科 iOS 出现，则将当前位置恢复到入学快照，再创建新快照学习 iOS 即可。

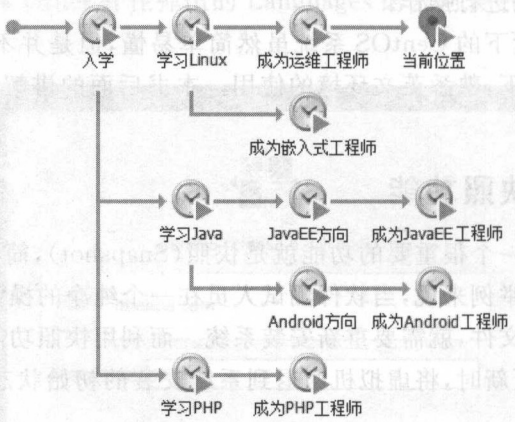


图 1-15 VMware 快照结构

1.3.4 VMware 克隆功能

Nginx 可以在集群环境下实现负载均衡调度，而集群环境需要多台服务器同时运行。在 VMware 中，要想搭建集群环境，就需要安装多台虚拟机，操作会比较麻烦。为了节省操作时间和硬盘空间，使用 VMware 的虚拟机克隆功能，可以很方便地将一个已经安装好的虚拟机克隆成多个。

下面通过具体的操作步骤演示如何进行虚拟机克隆。

(1) 在虚拟机关机的状态下，执行 VMware 菜单栏的“虚拟机”→“管理”→“克隆”操作。

(2) 在弹出克隆虚拟机向导后单击“下一步”按钮进入到“克隆源”页面，选择“虚拟机中的当前状态”，如图 1-16 所示。

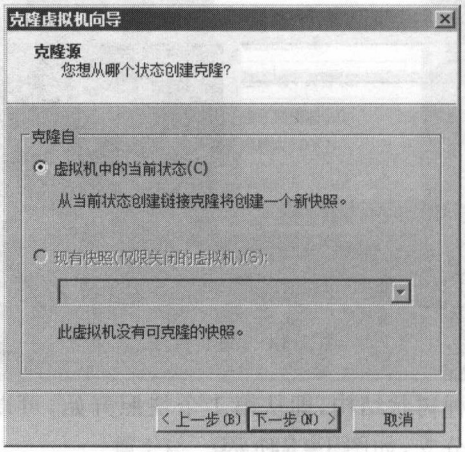


图 1-16 选择克隆源

值得一提的是，图 1-16 中“现有快照”的选项是灰色（不可用）的，这是因为当前虚拟机并没有创建过链接克隆，只有创建过才会产生快照。如果需要克隆出多个虚拟机时，第 1 次克隆会自动创建快照，后面几次克隆直接选择现有快照即可。

（3）选择克隆类型，如图 1-17 所示。链接克隆是一种方便快捷的克隆方法，推荐使用。

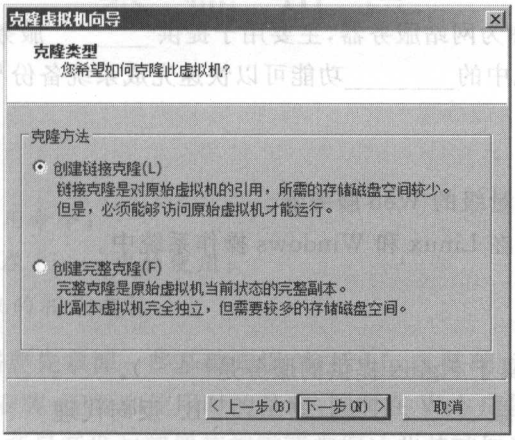


图 1-17　选择克隆类型

（4）配置克隆后的新虚拟机的名称和保存位置，选择一个较大的硬盘分区来保存即可。

（5）在完成虚拟机克隆后，即可使用新的虚拟机，如图 1-18 所示。被克隆虚拟机和新虚拟机的数据是互不影响的，但由于使用了链接克隆的方式，原虚拟机不能删除，否则克隆后的虚拟机将无法使用。

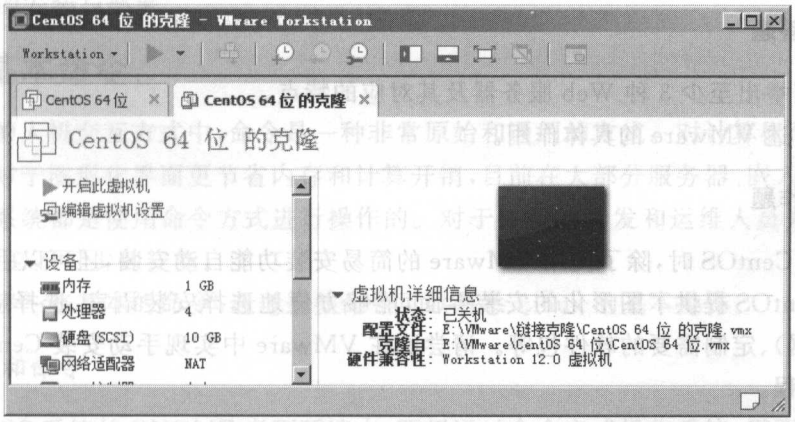


图 1-18　完成虚拟机克隆

本章小结

本章概括地介绍 Nginx 软件、Linux 系统和 VMware 虚拟机。通过本章的学习，读者可以了解 Nginx 的特点和主要应用，熟练使用虚拟机安装 Linux 系统，熟悉 Linux 桌面环境的基本操作，学会利用虚拟机的快照和克隆功能方便地管理实验环境。

课后练习

一、填空题

1. Web服务器被称为网站服务器，主要用于提供_____服务。
2. VMware虚拟机中的_____功能可以快速完成系统备份与还原。

二、判断题

1. Nginx是一个轻量级的Web服务器。（　　）
2. Nginx仅能运行在Linux和Windows操作系统中。（　　）

三、选择题

1. 下列选项中，不属于Nginx提供的服务是(　　)。
   A. HTTP服务器　　B. 反向代理
   C. 缓存　　D. 以上答案都不正确
2. 下列选项中，为高并发网站的应用设计的服务器是(　　)。
   A. Nginx　　B. Lighttpd　　C. Tomcat　　D. Apache
3. 单击虚拟机画面可将键盘和鼠标输入定向到虚拟机内，需要恢复时按(　　)键。
   A. Ctrl+Shift　　B. Ctrl+Shift+Alt
   C. Ctrl+Alt　　D. Shift+Alt

四、简答题

1. 请列举出至少3种Web服务器及其对应的特点。
2. 请简述VMware的具体作用。

五、操作题

在安装CentOS时，除了使用VMware的简易安装功能自动安装，还可以手动进行自定义安装。CentOS提供了图形化的安装界面，能够方便地选择安装语言、选择应用场景(如桌面、服务器)、定制需要的软件包等。请尝试在VMware中实现手动安装CentOS，体验完整的安装过程。

第 2 章
基 础 知 识

学习目标

- 掌握 Linux 的常用命令；
- 熟悉正则表达式在 Linux 中的使用；
- 了解 HTTP 协议的相关内容。

在学习 Nginx 前，需要先掌握一些基础知识，包括 Linux 基础、正则表达式和 HTTP 协议。这些知识是 Nginx 安装、配置和使用时的基础，同时在 Web 应用领域中也非常有用，例如服务器的搭建与维护、项目开发与部署等都需要掌握这些基础知识。

2.1 Linux 入门

Linux 是 Nginx 的主要运行平台，在使用 VMware 完成 Linux 的安装后，还需要掌握一些常用的操作命令，理解 Linux 的 shell、文件、用户、权限等机制以后，才能熟练操作 Linux，完成 Nginx 的安装与部署。

2.1.1 基本命令

在传统的人机交互方式中，命令是一种非常原始和通用的方式。对计算机而言，命令的字符界面相对于图形化界面更节省内存和计算开销，目前在大部分服务器、嵌入式设备上运行的 Linux 系统都是使用命令方式进行操作的。对于专业的开发和运维人员来说，熟练使用命令，更是必不可少的技能。

下面讲解一些基本的命令，通过这些命令可以熟悉 Linux 的基本操作。

1. 终端和命令

在 CentOS 系统的 GNOME 桌面环境中，要想通过命令方式操作系统，需要启动一个终端模拟器。在桌面上右击，在弹出的菜单中选择 Open in Terminal(在终端中打开)，就会打开一个如图 2-1 所示的虚拟终端的窗口。

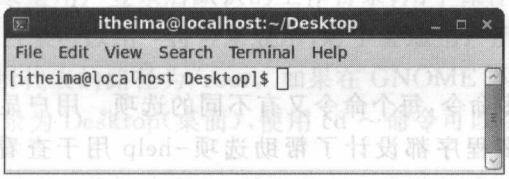


图 2-1 虚拟终端窗口

在这个窗口中可以输入命令，如输入 echo Hello World，然后按回车键，这行命令就会交给系统去执行，执行后的结果会在下面显示。当这行命令执行完成后就可以输入下一行命令。

```
[itheima@localhost Desktop]$echo Hello World
Hello World
[itheima@localhost Desktop]$
```

在以上示例中，echo 是系统中的一个程序，该程序用于接收输入的参数进行处理，处理完成后返回结果。echo 后面的 Hello World 是传递给程序的参数，下一行的 Hello World 是程序运行时输出的结果。当第 3 行输入命令的提示再次出现时，表示程序已经执行完成并退出。

终端中显示的[itheima@localhost Desktop]$ 描述的是当前正在以 itheima 用户的身份登录了名称为 localhost 的主机(也就是本地主机)，当前的工作目录是 Desktop(桌面)，最后的 $ 表示当前用户是一个普通用户，如果是超级用户 root 则显示为 #。

2. 命令格式

在 Linux 系统中，命令遵循的基本格式如下。

```
command [options] [arguments]
```

上述语法格式中，command 表示命令的名称，也可以理解为一个程序名；options 表示选项，定义了命令的执行特性；arguments 表示参数列表，通常用于表示命令作用的对象。另外，Linux 系统是严格区分大小写的，而 Windows 系统习惯上不区分大小写。因此，读者在输入 Linux 命令时一定要注意这个问题，养成在 Linux 系统中区分大小写的习惯。

下面以列出目录/home/itheima 下的所有文件为例进行演示：

```
[itheima@localhost Desktop]$ls -a /home/itheima
```

ls 命令用于查看某个目录下的文件列表，选项-a 表示显示所有文件，参数/home/itheima 表示该命令作用于这个目录。

每个命令的选项和参数是取决于程序如何设计的，有时候也可以省略。例如，省略-a 时，表示不显示隐藏的文件；省略/home/itheima 时，表示 ls 命令将作用于当前的工作目录。

值得一提的是，命令的选项有两种，分别为长选项和短选项。上述示例中的-a 为短选项，对应的长选项为--all。两种选项功能相同，区别在于多个短选项可以组合使用。例如，ls 命令还有一个常用选项-l 表示输出文件的详细信息，当同时用到两个选项时可以组合成-al，相当于-a -l。

3. 查看帮助

Linux 系统中有许多命令，每个命令又有不同的选项。用户显然不可能全部记住每个命令的选项。为此，许多程序都设计了帮助选项--help 用于查看使用说明。例如，通过 ls --help 可查看 ls 命令的帮助信息。除此之外，系统中还有一个帮助手册的命令 man(单词

manual 的缩写)。

以查看 echo 命令的帮助为例,执行 man echo 后的结果如图 2-2 所示。

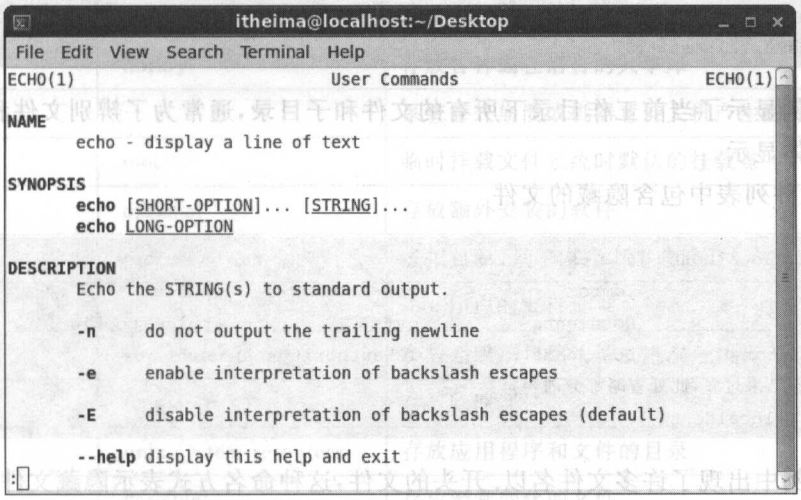


图 2-2 man 帮助命令

在显示帮助信息后,由于内容过多,需要用上下光标键进行滚动,按 PageUp 和 PageDown 键可以上下翻页,按 q 键退出程序。

4. 浏览目录

Linux 系统中浏览目录的操作,主要通过 ls、cd、pwd 3 个命令来完成,如表 2-1 所示。

表 2-1 浏览目录命令

命令名称	英文原意	说明
ls	list	显示目录中的文件列表
cd	change directory	切换工作目录
pwd	print working directory	显示当前的工作目录

为了更好地理解上述命令的使用,接下来通过一些具体的示例进行演示。

1) 显示当前工作目录

```
[itheima@localhost ~]$pwd
/home/itheima
```

上述示例中,pwd 命令显示了当前的工作目录为/home/itheima,该目录是当前用户 itheima 的家目录。家目录是用户登录后默认的工作目录,用于保存用户的个人文件。

另外,当前用户的家目录可以用~符号来表示,对于普通用户所代表的路径为"/home/用户名",对于 root 用户所代表的路径为/root。如果在 GNOME 环境下通过桌面右键方式打开虚拟终端,则工作目录为 Desktop(桌面),使用 cd ~命令可以切换到家目录。

2）显示当前工作目录下的文件列表

```
[itheima@localhost ~]$ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[itheima@localhost ~]$
```

上述示例显示了当前工作目录下所有的文件和子目录，通常为了辨别文件和目录，目录会以蓝色字体显示。

3）在文件列表中包含隐藏的文件

```
[itheima@localhost ~]$ls -a
.                .dmrc              .gtk-bookmarks    .pulse
..               Documents          .gvfs             .pulse-cookie
.bash_history    Downloads          .ICEauthority     .ssh
(由于输出结果过多，此处省略部分内容)
[itheima@localhost ~]$
```

上述示例中出现了许多文件名以.开头的文件，这种命名方式表示隐藏文件。在每个目录中都有两个特殊的文件.和..，分别表示当前目录和上级目录。

4）切换工作目录

```
[itheima@localhost ~]$cd Desktop
[itheima@localhost Desktop]$ls -a
.  ..
[itheima@localhost Desktop]$cd ..
[itheima@localhost ~]$
```

上述示例实现了将工作目录切换到 Desktop 子目录，然后查看目录中所有的文件，最后切换回上级目录。由此可见，通过 cd 和 ls 两个命令即可浏览系统中所有的文件和目录。另外，由于 Linux 的权限机制非常严格，有些目录需要 root 权限才可以查看，如果没有权限则会提示 Permission denied。

2.1.2 目录结构

熟悉 Windows 的用户通常习惯使用盘符路径来访问文件，如 C:\Windows。而 Linux 系统中并没有盘符的概念，而是通过根目录/表示所有文件的开始，并通过挂载(mount)的方式把所有硬盘分区都放置在“根”下的目录里。

使用命令 ls /可以查看根目录下的文件，表 2-2 列举了其中常用目录的说明。

表 2-2 Linux 目录结构说明

目录名称	英 文 原 意	说　明
bin	binary	二进制可执行文件目录(ls 等命令保存在此处)
boot	—	存放用于启动 Linux 系统的核心文件
dev	device	设备文件目录
etc	etcetera	存放系统的管理文件和配置文件

续表

目录名称	英 文 原 意	说　　明
home	—	存放普通用户的家目录
lib	library	存放各种编程语言的共享库
lost+found	—	系统意外崩溃或机器意外关机产生的一些文件碎片
mnt	mount	临时挂载文件系统时默认的挂载点
opt	optional	存放额外安装的软件
proc	process	虚拟目录，系统内存中的进程以文件形式的体现
root	—	root 用户的家目录
sbin	super user binary	存放超级用户使用的二进制可执行文件
tmp	temporary	存放临时文件
usr	unix system resources	存放应用程序和文件的目录
var	variable	存放经常变化的文件

表 2-2 中的目录都是 Linux 系统自动创建好的，每个目录都有特定的用途。Linux 目录结构体现了“一切皆文件”的设计理念，极大方便了开发人员使用统一的接口操作文件、设备、进程等。

在目录 usr 和 etc 中还有一些常用的目录，具体解释如下。

- /usr/bin：安装软件的二进制可执行文件目录。
- /usr/include：系统头文件(header files)的目录。
- /usr/local：存放管理员自行安装的软件。
- /usr/sbin：超级用户使用的二进制可执行文件的目录。
- /usr/src：源代码存放目录。
- /etc/passwd：保存系统中的用户。
- /etc/group：保存系统中的用户组。

从目录结构上看，/usr/local 和/usr 的内部结构非常相似，区别在于/usr 由 Linux 发行版的软件包管理器自动维护，而/usr/local 是管理员用户自行维护的。例如，用户通过软件包管理器安装一个软件，则软件自动安装到/usr 中；而用户手动安装软件，则安装到/usr/local 中。

2.1.3 shell 和终端

1. shell

在学习 Linux 系统时，理解 shell 的概念非常重要。shell 是一种包裹在系统内核之外的“壳”，其功能主要用于解释用户输入的命令，加以执行。例如，用户在终端中输入命令 ls -a，则这条命令就会发送给 shell 程序，然后 shell 在/bin 目录下找到 ls 命令，执行该命令并传入选项-a。

用户登录后执行 echo $SHELL 可以查看当前使用的 shell 程序，如下所示。

```
[itheima@localhost ~]$echo $SHELL
/bin/bash
[itheima@localhost ~]$
```

上述执行结果中的/bin/bash 就是 shell 程序 Bash 的保存路径。Bash(GNU Bourne-Again Shell)是一个为 GNU 计划编写的 Unix shell,目前大多数 Linux 发行版的默认 shell 都使用 Bash。

Bash 除了能够执行基本的命令,还支持编程,提供如变量、运算、条件判断、循环语句、函数等语法功能。用户可以将一系列使用 shell 编程的代码保存到一个 shell 脚本文件中(通常使用 sh 扩展名),能够像普通程序一样执行。

2. 终端

终端(Terminal)在传统意义上讲是指终端设备,是计算机网络中处于最外围的设备,主要用于用户向计算机输入信息,以及处理结果的返回。举例来说,用户使用手机、计算机等设备访问互联网,浏览由网络中的服务器提供的信息时,就把用户使用的这些设备称为终端。

使用终端方式访问计算机可以实现多用户、多任务地工作,因为每个终端都有显示屏和输入设备。而若只有一台计算机时(如个人计算机、未部署的服务器),则计算机本身也可以称之为终端,或控制台终端(console)。在 GNOME 桌面环境中,可以打开多个虚拟终端进行操作,但若没有安装桌面,还可以使用系统本身提供的虚拟控制台。

Linux 系统提供 6 个虚拟控制台终端,分别是 tty1、tty2、tty3、…、tty6。其中 tty1 是开机后自动进入的,当安装 GNOME 桌面时,tty1 将显示桌面环境,而其他终端仍然是字符界面。在 GNOME 环境下按 Ctrl+Alt+F2~F6 键可以切换到其他终端。以 tty2 为例,按 Ctrl+Alt+F2 键切换到 tty2,然后登录系统,使用 tty 命令查看当前所在的终端,运行结果如图 2-3 所示。

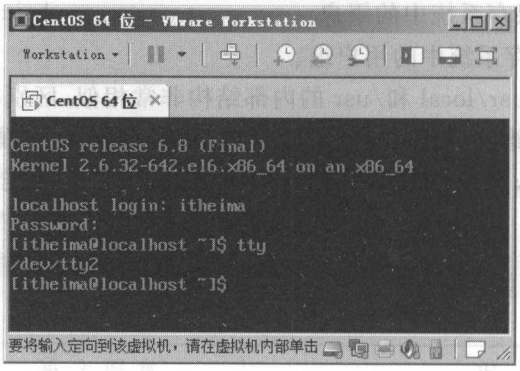


图 2-3 查看当前终端

在非桌面环境下,还可以按 Alt+F1~F6 键切换到其他虚拟终端。例如,在 tty2 终端环境下,可以按 Alt+F1 键返回 tty1 的桌面环境。

值得一提的是,若在 GNOME 桌面环境中打开虚拟终端,则用 tty 命令看到的终端设备为 pts 终端,路径为/dev/pts/0。pts 和 tty 的区别在于,pts 是一种用于远程登录的终端,如

果通过远程连接(如 SSH)的方式登录系统,则会进入 pts 终端中。关于远程连接的相关课程会在后面的章节中进行讲解。

3. 其他命令和操作

在使用终端访问 Linux 系统时,还有一些操作可以方便使用,具体如下。

1) 清屏

当在终端中执行过多次命令后,屏幕中就会累积大量的历史操作信息。如果想要清除这些信息,可以用 clear 命令。当 clear 执行后,当前屏幕中所有的内容将会被清空。但如果是 GNOME 的虚拟终端或远程连接(如 SSH)的方式登录系统,则仍然可以向上滚动查看历史信息。另外,使用快捷键 Ctrl+L 也可以清屏。

2) 自动补全路径

当需要在一行命令中输入一个实际存在的路径时,可以用 Tab 键实现自动补全。例如,在 ls 命令中输入一个不完整的路径 ls /home/it,此时按 Tab 键,就会自动将末尾的 it 补全为 itheima(前提是该文件实际存在),如果在同一个目录下有多个以 it 开始的文件,则不会自动补全,但会将符合条件的文件列出来。

3) 历史命令切换

经常使用命令的用户可能会遇到这样一种情况,前面输入一段很长的命令由于其中一个字符写错导致整个命令无法执行,又要重新输入一遍,相当麻烦。对于这个问题有个简单的方法,就是按上下光标键切换历史命令。在一个终端里执行过的命令是被自动记忆的,通过 history 命令可以查看历史记录。

4) 使用变量

变量是指用一个 $ 开始的标识符表示一个可变的值,通常用于 shell 编程环境中。Linux 系统中有许多预定义变量用于访问系统环境信息,用户也可以自己定义变量,普通变量的生命周期为终端或 shell 脚本内。下面通过具体操作步骤演示如何访问变量、如何定义变量。

```
[itheima@localhost ~]$echo $HOME
/home/itheima
[itheima@localhost ~]$echo $HOSTNAME
localhost.localdomain
[itheima@localhost ~]$hello="Hello World"
[itheima@localhost ~]$echo $hello
Hello World
```

上述示例首先通过 echo 输出预定义变量 $HOME 和 $HOSTNAME 的值,然后又定义一个 $hello 变量并赋值为“Hello World”,最后输出 $hello 的值。其中,$HOME 表示当前用户家目录的路径,$HOSTNAME 表示当前系统的主机名称。

5) 使用反引号

反引号“`”的作用是将反引号内的命令优先执行,然后将原本输出到终端的执行结果赋予原来的命令中使用。具体示例如下。

```
[itheima@localhost ~]$test=`ls`
[itheima@localhost ~]$echo $test
Desktop Documents Downloads Music Pictures Public Templates Videos
[itheima@localhost ~]$echo Date=`date`
Date=Mon Oct 17 03:18:17 PDT 2016
```

从上述示例中可以看出，通过反引号包裹的 ls 和 date 命令已经执行。其中，date 命令用于获取当前的系统时间。

6）字符串转义问题

在输入命令时，若参数值中包含空格等特殊字符时，会遇到识别错误的问题，这就需要利用双引号或单引号进行包裹。双引号定界符会自动解析字符串，包括读取变量值、执行反引号中的命令，而单引号定界符内字符串则会原样输出，且其内部不能出现单引号。

若在双引号定界符中出现“”“`”“\”这些特殊字符时，可以在字符前加一个“\”转义，以避免被解析。下面通过示例演示两种定界符的区别。

```
[itheima@localhost ~]$test=Hello
[itheima@localhost ~]$echo "$test World \" \\"
Hello World " \
[itheima@localhost ~]$echo '$test World \" \\'
$test World \" \\
```

从上述示例可以看出，在双引号字符串中，变量 $test 的值被取出来，“\”“被转义成“””，“\\”被转义成“\”，而在单引号字符串中，输出结果是原样显示的。

7）退出终端

当用户需要离开终端时，可以使用 exit 命令。需要注意的是，同一个用户可以同时登录多个终端，exit 命令仅用于退出当前终端，而非在所有的终端中退出同一个用户。

在 Linux 终端环境下，执行 exit 命令会退出当前登录的用户，然后出现登录界面，可以重新输入用户名和密码进行登录。对于 GNOME 环境下的虚拟终端，exit 仅用于退出虚拟终端，而非退出 GNOME 环境。

4. 运行级别

运行级别是指当前操作系统正在运行的功能级别，通过数字 0～6 表示 7 个级别，每个级别具有不同的功能。这些级别定义在/etc/inittab 文件中，每个级别运行的服务程序存放在/etc/rc.d 目录中。

关于 Linux 的 7 种运行级别可参照表 2-3 所示。

表 2-3 运行级别

级别	说　明	备　注
0	系统停机模式	默认运行级别不能设置为 0，否则无法正常启动
1	单用户模式	仅用于 root 用户维护系统，类似于 Windows 中的安全模式
2	多用户模式	没有 NFS(Network File System，网络文件系统)支持

续表

级别	说　明	备　注
3	多用户模式	有 NFS 的完整多用户模式，登录后进入控制台
4	未使用模式	系统未使用的保留模式
5	图形化模式	类似于运行级别 3，登录后进入桌面环境
6	重启模式	默认运行级别不能设置为 6，否则无法正常启动

当前系统的运行级别可以在 root 用户身份下通过 init 命令进行切换，示例如下。

```
[itheima@localhost ~]$su root
Password:
[root@localhost itheima]#init 3
```

在上述示例中，su 命令用于切换当前用户身份，切换成 root 时会提示输入密码，且在输入时没有回显提示。VMware 简易安装方式设置的 root 密码与 itheima 用户相同。在登录 root 成功后执行 init 3 命令，即可切换到控制台模式。而在控制台中执行 init 5 命令，就会进入图形化模式。

通过 init 命令可以实现系统的关机或重启，执行 init 0 关机，init 6 重启。除此之外，也可以使用命令 poweroff 实现关机，reboot 实现重启。

2.1.4 文件管理

文件管理是操作系统的基本功能，主要包括对文件和目录的创建、查看、修改、删除 4 类操作，下面将结合示例，对这 4 类操作依次进行讲解。

1. 文件创建

1）touch 方式创建文件

touch 命令的主要功能是将已存在文件的时间属性更新为当前系统时间，若指定的文件不存在，则会创建一个新的空文件。具体示例如下。

```
[itheima@localhost ~]$touch hello.txt
[itheima@localhost ~]$ls
hello.txt
```

通过 ls 命令的选项-l 可以查看文件的修改时间，ls -l 还可以缩写为 ll 命令。下面的示例演示了如何查看文件的修改时间。

```
[itheima@localhost ~]$ll hello.txt
-rw-rw-r--. 1 itheima itheima 0 Oct 18 00:05 hello.txt
[itheima@localhost ~]$touch hello.txt
[itheima@localhost ~]$ll hello.txt
-rw-rw-r--. 1 itheima itheima 0 Oct 18 00:06 hello.txt
```

从上述示例可以看出，hello. txt 文件的修改时间从 00:05 更新为 00:06。

2）重定向方式创建文件

Linux 系统中默认的标准输入输出设备分别是键盘和显示器，且支持对输入和输出进行重定向。输入重定向能够将某个文件的内容作为输入信息，输出重定向能够将原本输出到屏幕的信息输出到文件中保存。因此，通过输出重定向可以实现创建文件并写入内容。

接下来通过>、>>两种符号实现输出重定向，其中>是覆盖方式（文件存在时覆盖文件），>>是追加方式（文件存在时向内追加文本），示例如下。

```
[itheima@localhost ~]$echo Hello World >hello.txt
[itheima@localhost ~]$echo itheima.com >>hello.txt
[itheima@localhost ~]$cat hello.txt
Hello World
itheima.com
```

上述示例中，cat 命令用于读取文本文件的内容并显示，从输出结果可知，利用输出重定向成功向文本文件 hello.txt 中写入两行文本信息。

3）创建目录

使用 mkdir 命令可以创建目录。该命令的选项-p 能够实现自动创建路径中不存在的目录，若省略该选项，则新创建目录的上级目录必须是已经存在的。

接下来实现在当前目录下创建 itheima/study 两层目录，示例如下。

```
[itheima@localhost ~]$mkdir -p itheima/study
[itheima@localhost ~]$cd itheima/study
[itheima@localhost study]$pwd
/home/itheima/itheima/study
```

上述示例中，mkdir 命令添加了选项-p，表示在创建 study 目录时，如果上级目录 itheima 不存在，则先创建 itheima 后再创建 study 目录。若省略该选项且 itheima 目录不存在，该命令会执行失败。

2. 文件查看与搜索

1）查看文件

在 Linux 中查看文件的命令有很多，其中常用的有 cat 和 less。cat 命令用于将文件全部读取并显示出来，在前面已经用过；less 命令可以在打开文件后随意浏览，支持上下翻页。

less 命令在打开文件后的操作和 man 命令类似：用上下光标键滚动屏幕内容、PageUp 和 PageDown 键翻页、q 键退出。以查看某个文件为例，执行 less /etc/service 命令，运行结果如图 2-4 所示。

2）搜索文件

通过 find 命令可以根据搜索条件到指定路径下搜索文件，支持递归搜索子目录。其语法格式如下。

```
find 搜索路径 [选项] 搜索关键字
```

find 命令的选项有很多，其常用的有-name、-size 和-user，分别用于根据文件名称查找、

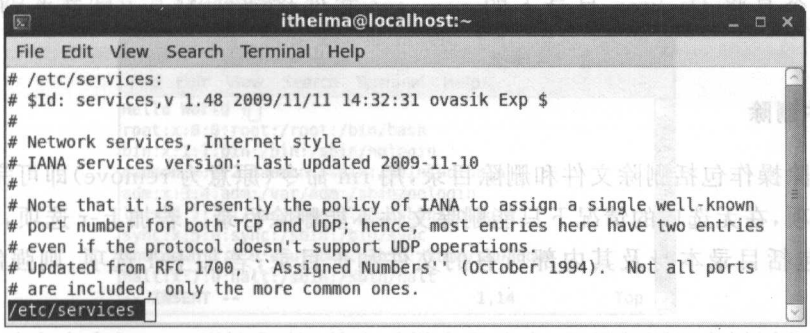


图 2-4 less 命令查看文件

根据文件大小查找、根据文件所有者查找。下面以选项-name 为例，在根目录下搜索所有文件名以 bas 开始的文件。

```
[itheima@localhost ~]$ find /bin -name "bas*"
/bin/basename
/bin/bash
```

从上述示例可知，find 命令成功搜索到两个符合条件的文件。搜索文件名时可以用通配符 * 进行模糊搜索，若将示例中的搜索条件改为 *.sh，则表示查找文件名以.sh 结束的文件。

3. 文件修改

文件修改操作包括重命名和移动路径，可以用 mv 命令完成，其语法格式如下。

```
mv 源文件或目录 目标文件或目录
```

由于 mv 命令的用法非常灵活，下面利用具体的示例进行操作演示。

1）重命名文件

当源文件或目录存在，目标文件或目录不存在时，执行重命名操作。

```
[itheima@localhost ~]$ mv hello.txt new.txt
```

上述命令将源文件 hello.txt 重命名为 new.txt。

2）移动文件

当目标文件是一个已经存在的目录时，会将源文件或目录移动到目标目录中。

```
[itheima@localhost ~]$ mv new.txt Desktop
```

上述命令将文件 new.txt 移动到 Desktop 目录中。

3）移动并重命名文件

当源文件和目标文件不在同一个目录中时，如果目标文件不存在，则执行移动并重命名操作。

```
[itheima@localhost ~]$ mv Desktop/new.txt Music/hello.txt
```

上述命令是将 Desktop 目录下的 new.txt 文件移动到 Music 目录下，并重命名为 hello.txt。

4. 文件删除

文件删除操作包括删除文件和删除目录，用 rm 命令(原意为 remove)即可完成。rm 命令有许多选项，在无选项的情况下只能删除文件不能删除目录。若加上-r 选项，则递归删除指定目录，包括目录本身及其内部所有的文件和子目录；若加上-f 选项，则强制执行删除操作。

下面的示例创建了测试目录，在目录中增加文件，然后利用 rm -rf 命令全部删除。

```
[itheima@localhost ~]$mkdir -p test/aaa/bbb
[itheima@localhost ~]$echo test >test/aaa/bbb/1.txt
[itheima@localhost ~]$rm -rf test
[itheima@localhost ~]$ls test
ls: cannot access test: No such file or directory
```

在上述操作中，当完成 test 目录的删除操作后，再使用 ls 命令查看目录时，提示没有这个文件或目录，说明 test 目录和其内部所有的文件和子目录已经被全部删除。

2.1.5 vi 编辑器

vi 编辑器是 Linux 系统下最基本的编辑器，它没有图形界面，非常高效和实用。vi 编辑器具有 3 种工作模式，分别是命令模式(command mode)、插入模式(insert mode)和底行模式(last line mode)。

在学习这 3 种模式之前，首先学习如何使用 vi 编辑器打开要编辑的文件。下面将系统中的/etc/passwd 文件复制到用户目录，然后用 vi 编辑器打开它，具体操作如下。

```
[itheima@localhost ~]$cp /etc/passwd passwd
[itheima@localhost ~]$vi passwd
```

上述命令执行后，即可打开用户家目录中的 passwd 文件。另外，若 vi 命令后面参数指定的文件不存在，则编辑器启动后显示的是空白内容，在编辑器中执行保存操作后即可创建出一个新文件。

1. 快速上手

vi 编辑器在启动后默认是命令模式，该模式下许多按键都有特定的功能，不能直接编写内容。按 i 键可以进入插入模式，屏幕左下角显示-- INSERT --，这时候就可以用键盘直接编写文件内容，如图 2-5 所示。按 Esc 键可以从插入模式返回命令模式。

当文本编写完成后，若要保存文件或退出编辑器，可以在切换到命令模式后，按“:”进入底行模式，然后可以在屏幕左下角输入命令按回车键执行。其中常用的命令有：

- :q 退出 vi 编辑器(若改动过文件内容未保存则不允许退出)；
- :q! 强制退出 vi 编辑器不保存文件；
- :w 保存编辑后的文件；

- :wq 保存文件并退出编辑器。

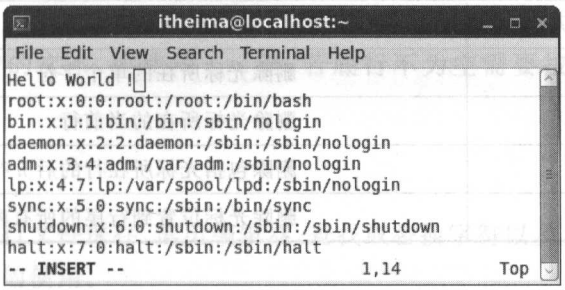


图 2-5　编辑文本内容

2. 命令模式

在命令模式下，主要功能是通过键盘控制光标的移动、文本内容的复制、粘贴、删除等。下面分别介绍其相关命令。

1）光标移动

光标的移动可分为 6 个常用的级别，分别为字符级、行级、单词级、段落级、屏幕级和文档级。各个级别中的相关按键及其含义如表 2-4 所示。

表 2-4　光标移动操作

级别	操作符	说　明
字符级	“左键”或字母 h	使光标向字符的左边移动
	“右键”或字母 l	使光标向字符的右边移动
行级	“上键”或字母 k	使光标移动到上一行
	“下键”或字母 j	使光标移动到下一行
	符号 $	使光标移动到当前行尾
	数字 0	使光标移动到当前行首
单词级	字母 w	使光标移动到下一个单词的首字母
	字母 e	使光标移动到本单词的尾字母
	字母 b	使光标移动到本单词的首字母
段落级	符号}	使光标移至段落开头
	符号{	使光标移至段落结尾
屏幕级	字母 H	使光标移至屏幕首部
	字母 L	使光标移至屏幕尾部
文档级	字母 G	使光标移至文档尾行
	n+G	使光标移至文档的第 n 行(如 5G 移到第 5 行)

2）删除

当需要删除文件中的某些内容时，具体操作如表 2-5 所示。

表 2-5 删除操作

操作符	说　明
字母 x	删除光标所在的单个字符
字母 dd	删除光标所在的当前行
n+dd	删除包括光标所在行的后 n 行内容
d+$	删除光标位置到行尾的所有内容

3）复制和粘贴

当需要复制、粘贴文件中的某些内容时，具体操作如表 2-6 所示。

表 2-6 删除操作

操作符	说　明
字母 yy	复制光标所在当前行
n+yy	复制包括光标所在行的后边 n 行内容
y+e	从光标所在位置开始复制直到当前单词结尾
y+$	从光标所在位置开始复制直到当前行结尾
y+{	从光标所在位置开始复制直到当前段落开始的位置
P	将复制的内容粘贴到光标所在位置

除了以上 3 类，在命令模式下还有如下几种常见的操作：

- 字母 u：撤销命令。
- 符号“.”：重复执行上一次命令。
- 字母 J：合并两行内容。
- r+字符：快速替换光标所在字符。

3. 模式切换

在命令模式下可以切换到插入模式或底行模式，下面详细介绍如何进行模式切换。

1）命令模式切换插入模式

前面讲过用 i 键可以切换到插入模式，在切换时光标位置和内容没有发生变化。除此之外，还有许多按键也可以进入插入模式，如表 2-7 所示。

表 2-7 切换到插入模式

操作符	切换插入模式并执行操作
字母 a	光标向后移动一位
字母 A	光标移动到当前行末尾
字母 I	光标移动到当前行行首
字母 s	删除光标所在字符
字母 S	删除光标所在行
字母 o	在当前行之下新增一行
字母 O	在当前行之上新增一行

2) 命令模式切换底行模式

在命令模式下输入“:”或“/”,可进入底行模式。若想从底行模式返回到命令模式,可以用退格键删除底行中的文本,或使用 Esc 键,若底行不为空需要按两次 Esc 清空底行并返回。

4. 底行模式

底行模式可以进行文件保存、退出编辑器、查找或替换字符以及显示行号等操作。下面分别介绍其常用命令的使用。

(1) “:set number”或“:set nu”: 显示行号。

(2) “:set nonumber”或“:set nonu”: 取消行号显示。

(3) “:n”: 使光标跳转到第 n 行。

(4) “:/xx”或“/xx”: 在文件中查找 xx 内容。若查找结果不为空,按 n 键向下查找,N 键向上查找。

(5) 内容替换操作,具体如表 2-8 所示。

表 2-8 内容替换操作

操 作 符	说 明
:s/被替换内容/替换内容/	替换光标所在行的第一个目标
:s/被替换内容/替换内容/g	替换光标所在行的全部目标
:%s/被替换内容/替换内容/g	替换整个文档中的全部目标
:%s/被替换内容/替换内容/gc	替换整个文档中的全部目标,每替换一个内容都有提示

(6) 保存与退出操作,在“快速上手”阶段已经讲过,这里不再重复。

2.1.6 用户和权限

Linux 中设定了用户和权限的概念,在使用操作系统时,必须先以某个用户的身份登录系统。用户的账号分为多种类型,不同账号拥有的权限、担任的角色也各不相同。例如,拥有系统全部操作权限的超级用户 root、权限受到一定限制的普通用户,以及仅用于维持系统或某个程序正常运行的程序用户。下面对用户和权限进行详细讲解。

1. 用户管理

Linux 是一个多用户、多任务的操作系统,在一台 Linux 主机上,可能同时登录多名使用者。为了对这些使用者的状态进行跟踪,对其可访问的资源进行控制,就需要对用户账号机制进行管理。

1) 添加用户

添加用户使用 useradd 命令完成,普通用户没有执行该命令的权限,需要 root 用户才能执行。useradd 的基本语法格式如下。

```
useradd 选项 用户名
```

通过 useradd 的选项可以设置新用户的详细信息，常用选项如表 2-9 所示。

表 2-9 useradd 常用选项

选项	省略时的默认值	说　明
-d	/home/用户名	指定用户的家目录
-c	空	指定用户的备注文字
-g	自动创建同名用户组	指定用户所属的基本组
-G	无	指定用户所属的附加组
-r	否	创建系统账号
-s	/bin/bash	指定用户登录的 shell 程序
-u	从 500 开始递增	指定用户 ID

需要注意的是，Linux 中有许多系统级的用户和用户组，占用了一定数量的 ID，因此 useradd 创建的普通用户的 ID 和用户组的 ID 都将从 500 开始递增。如果加上选项-r，则新用户 ID 小于 500。

下面通过具体的示例演示如何创建新用户。

```
#直接创建新用户 bxg
[root@localhost ~]#useradd bxg
#创建新用户 bxg2,指定其备注信息和所属组
[root@localhost ~]#useradd -c ng.test -g itheima bxg2
#创建新用户 bxg3,指定其用户 ID 和所属组 ID
[root@localhost ~]#useradd -u 678 -g 500 bxg3
```

当用户创建后，在/etc/passwd 文件中会添加新用户的记录。通过查看该文件的末尾 3 行，可以看到以上新建的 3 个新用户的信息，查询结果如下。

```
[root@localhost ~]#tail -3 /etc/passwd
bxg:x:501:501::/home/bxg:/bin/bash
bxg2:x:502:500:ng.test:/home/bxg2:/bin/bash
bxg3:x:678:500::/home/bxg3:/bin/bash
```

在上述示例中，tail 命令用于查看文件末尾指定行数的内容。passwd 文件的每一行是一个用户账号的信息，每个信息用“:”分隔。这些信息共 7 项，分别是用户名、密码的占位、用户 ID、用户组 ID、备注信息、家目录、登录 shell。

在新用户创建好以后，可以用“su 用户名”切换当前用户身份。但是由于这些用户并没有设置密码，尚处于锁定状态，因此无法登录系统。下面将会讲解如何设置用户密码。

2）设置密码

设置用户密码的命令是 passwd，该命令的常用选项如表 2-10 所示。这些选项只有 root 身份可以用，普通用户执行 passwd 只能修改自己的密码。

表 2-10 passwd 常用选项

选项	说 明
-l	锁定密码，锁定后无法登录(新用户默认锁定，更改密码后自动解锁)
-u	解锁密码
-d	删除密码(删除后登录时将无须输入密码)
-S	列出密码相关信息

下面演示如何使用 passwd 为用户 bxg 设置密码。

```
[root@localhost ~]#passwd bxg
Changing password for user bxg.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

在输入密码时，Linux 会检查密码的强度。普通用户的密码要求尽量复杂(至少 6 位、包含字母和数字)，如果是常见的弱密码或与用户名相同，会出现 BAD PASSWORD 提示。

用户的密码保存在/etc/shadow 文件中，且密码是加密存储的，直接查看该文件的内容无法获知密码原文。为了安全，普通用户没有查看该文件的权限。

3) 删除用户

删除用户使用 userdel 命令，需要 root 权限。该命令的选项-r 用于在删除用户的同时删除与其相关的所有文件，如用户的家目录；-f 用于强制删除用户，即使该用户已经登录。

下面演示如何删除用户 bxg，并删除与用户相关的文件。

```
[root@localhost ~]#userdel -r bxg
userdel: user bxg is currently used by process 3860
[root@localhost ~]#userdel -rf bxg
userdel: user bxg is currently used by process 3860
[root@localhost ~]#userdel bxg
userdel: user 'bxg' does not exist
```

上述示例使用了 3 次 userdel 命令，第 1 次在删除时发现正在被进程 3860 占用，用户 bxg 正在登录时会出现此情况；第 2 次添加-f 选项强制删除账号，此时 bxg 用户和相关文件已经被删除；第 3 次再用 userdel 删除 bxg 用户，就会提示该用户不存在。

2. 用户组管理

系统中的每个用户必须属于一个基本用户组，可拥有多个附加用户组。使用 useradd 添加用户时，如果没有指定用户组，默认会为用户创建一个同名的用户组作为用户的基本组。使用 userdel 删除用户时，如果同名用户组中没有其他用户则会被自动删除，但若同名用户组非该用户基本组的情况除外。

用户组除了由程序自动创建和删除，root 用户也可以手动管理。

1) 添加用户组

添加用户组使用 groupadd 命令，其选项-g 用于指定组 ID，省略时自动使用从 500 开始

的递增值；选项-r表示创建系统用户组，其组ID小于500。

下面进行案例演示。创建group1和group2两个用户组，并设置group1的ID为550，具体如下。

```
[root@localhost ~]#groupadd -g 550 group1
[root@localhost ~]#groupadd group2
```

创建后使用tail命令查看/etc/group文件中保存的用户组结果，输出信息如下。

```
[root@localhost ~]#tail -3 /etc/group
group1:x:550:
group2:x:551:
```

从上述结果可以看出，新用户组group2的组ID为551，该值在创建用户组时自动生成，是用户组中ID的最大值加1的结果。

2）删除用户组

删除用户组使用groupdel命令，下面演示如何删除group2用户组。

```
[root@localhost ~]#groupdel group2
```

需要注意的是，如果要删除的组是某个用户的基本组则无法删除，除非删除用户或改变用户的基本组。

3）修改用户组

使用groupmod命令可以修改用户组的ID和组名，usermod命令可以修改用户名和用户所属用户组。下面通过具体案例演示这两个命令的使用。

```
#修改用户组group1的组ID为555
[root@localhost ~]#groupmod -g 555 group1
#修改用户组group1的组ID为666,并更名为group2
[root@localhost ~]#groupmod -g 666 -n group2 group1
#修改用户bxg所属基本组为group2
[root@localhost ~]#usermod -g group2 bxg
#修改用户bxg的附加组列表(多个组用“,”分隔)
[root@localhost ~]#usermod -G itheima,test1 bxg
#向用户bxg的附加组列表中追加组(多个组用“,”分隔)
[root@localhost ~]#usermod -a -G test2,test3 bxg
```

若用户已经登录系统，在修改用户的基本组或附加组后，只有下次登录后才会生效。通过id或groups命令可以查看当前用户或指定用户属于哪些用户组。具体示例如下。

```
[root@localhost ~]#id itheima
uid=500(itheima)gid=500(itheima)  groups=500(itheima)
[root@localhost ~]#groups
root
```

3. 文件权限管理

前面讲解了Linux系统的用户和用户组机制，这些机制可以用于对文件权限的控制。

Linux 中的用户对于文件有 3 种操作身份：文件所有者(owner)、文件所属组(group)和其他人(other)，详细说明参照表 2-11。每个身份对文件的操作由 3 种权限控制，分别是读取(read)、写入(write)和执行(execute)，详细说明参照表 2-12。

表 2-11 操作身份说明

用户身份	字符代号	说 明
文件所有者	u	文件所有者对文件的权限(通常是用户自己创建的文件)
文件所属组	g	文件所属组中的用户对文件的权限
其他人	o	文件所有者、所属组以外的其他人对文件的权限

表 2-12 权限控制说明

权限	字符代号	文 件	目 录
读取	r	可查看文件内容	可以列出目录中的内容
写入	w	可修改文件内容	可以在目录中创建、删除文件
执行	x	可执行该文件	可以进入目录

1) 查看文件权限

在 ll 命令列出的文件列表详细信息中包含了文件权限的信息，具体的示例如下。

```
[root@localhost ~]#ll /etc
total 1824
drwxr-xr-x.  3 root root   4096 Oct 12 07:48 abrt
drwxr-xr-x.  4 root root   4096 Oct 12 07:54 acpi
-rw-r--r--. 1 root root     45 Oct 17 23:06 adjtime
-rw-r--r--. 1 root root   1512 Jan 12 2010 aliases
⋮
```

上述输出结果共划分 7 个字段列表，其表示的信息依次为文件类型与权限、文件硬链接数、文件所有者、文件所属组、文件所占空间、文件修改时间、文件名。

文件类型和权限由 10 个字符组成，如 drwxr-xr-x。第 1 个字符代表文件类型，d 表示目录，-表示普通文件。第 2～10 位字符用于权限控制，每 3 位为 1 组，依次代表文件所有者、文件所属组、其他人对文件的权限，每组的最高权限为 rwx(可读、可写、可执行)，最低权限为---(完全没有权限)。

如果用户创建一个新文件，则新文件的所有者就是该用户，所属组是用户的基本组。

2) 更改文件权限

利用 chmod 命令可以更改文件权限，有字母和数字两种修改方法。如果作用于目录，加上选项-R 可以递归修改目录中的子目录和文件，否则只修改目录本身。

下面演示 chmod 字母权限表示方式的修改示例，具体如下。

```
#将权限设置为“rwxrwxrwx”
[root@localhost ~]#chmod u+rwx,g+rwx,o+rwx test.txt
#将“rwxrwxrwx”改为“rwxr-xr--”
[root@localhost ~]#chmod g-w,o-wx test.txt
#将权限设置为“rw-rw-rw-”
[root@localhost ~]#chmod u=rw-,g=rw-,o=rw-test.txt
#将“rw-rw-rw-”改为“r--r--r--”
[root@localhost ~]#chmod -w test.txt
#将“r--r--r--”改为“rwxr-----”
[root@localhost ~]#chmod u+wx,o-r test.txt
```

从上述示例可以看出，chmod 字母方式修改权限非常灵活，通过 u、g、o 指定文件所有者、所属组和其他人 3 种身份，+、-、=分别表示添加、取消、指定权限。

数字修改方式则通过数字表示权限，1 表示执行，2 表示写入，4 表示读取。通过数字相加可以将 3 种权限叠加，最高权限为 7(1+2+4)，最低权限为 0。具体示例如下。

```
#将权限设置为“rwxrwxr--”
[root@localhost ~]#chmod 774 test.txt
#将权限设置为“rw-r--r--”
[root@localhost ~]#chmod 644 test.txt
#将权限设置为“----------”
[root@localhost ~]#chmod 0 test.txt
```

从上述示例可知，u、g、o 3 种身份分别由 3 位数字表示，如 777 表示所有身份都拥有最高权限，0 表示所有身份都没有权限。

3) 更改文件所有者和所属组

使用 chown 命令可以更改文件所有者和所属组，若只更改所属组也可以使用 chgrp 命令。涉及目录时，这两个命令同样支持-R 选项递归处理。具体示例如下。

```
#将文件的所有者改为 itheima
[root@localhost ~]#chown itheima test.txt
#将文件的所属组改为 itheima
[root@localhost ~]#chgrp itheima test.txt
#将文件的所有者改为 bxg,所属组改为 itheima
[root@localhost ~]#chown bxg:itheima test.txt
```

为了验证权限控制是否有效，接下来通过一个完整的示例进行演示。

(1) 创建测试用户 bxg1、bxg2、bxg3，以及测试目录/home/test。

```
[root@localhost ~]#useradd bxg1
[root@localhost ~]#useradd bxg2
[root@localhost ~]#useradd bxg3
[root@localhost ~]#cd /home
[root@localhost home]#mkdir test
```

(2) 将测试目录的所有者改为 bxg1，所属组改为 bxg2，权限改为 rwxr-----。

```
[root@localhost home]#chown bxg1:bxg2 test
[root@localhost home]#chmod 740 test
```

(3) 执行 su bxg1 切换到用户 bxg1,利用 ls、cd、touch 测试读取、执行、写入权限。

```
[bxg1@localhost home]#ls test
[bxg1@localhost home]#cd test
[bxg1@localhost test]#touch 1.txt
[bxg1@localhost test]#rm 1.txt
```

(4) 测试用户 bxg2 的权限(先执行 exit 返回 root 再执行 su bxg2 完成切换)。

```
[bxg2@localhost home]$ls test
[bxg2@localhost home]$cd test
bash: cd: test: Permission denied
[bxg2@localhost home]$touch test/2.txt
touch: cannot touch `test/2.txt': Permission denied
```

(5) 测试用户 bxg3 的权限。

```
[bxg3@localhost home]$ls test
ls: cannot open directory test: Permission denied
[bxg3@localhost home]$cd test
bash: cd: test: Permission denied
[bxg3@localhost home]$touch test/3.txt
touch: cannot touch `test/3.txt': Permission denied
```

通过以上测试结果可以看出,文件所有者 bxg1 拥有读、写、执行权限,文件所属组的用户 bxg2 仅拥有读取权限(ls 成功,cd 和 touch 失败),"其他人"身份的 bxg3 没有任何权限。

需要注意的是,"文件所属组"与"文件所有者所属的用户组"无关,可以修改成其他用户组,此时文件所有者和文件所属组中的用户将同时具有权限。由于一个用户可以拥有一个基本组和多个附加组,只要其中一个组是文件所属组,就对该文件拥有权限。

4. 查看登录的用户

从管理员角度来说,目前系统中登录多少用户,分别使用哪种类型的终端,占用多少 CPU 资源,这些信息都非常重要。正如 Windows 系统通过"任务管理器"查询系统运行状态,在 Linux 系统中可以用 w 命令查询,无论是 root 还是普通用户都可以执行此命令。

在登录几个测试用户后,w 命令的输出结果如下所示。

```
[root@localhost ~]#w
01:35:53 up 5 min,  4 users,  load average: 0.63, 0.37, 0.17
USER     TTY      FROM     LOGIN@   IDLE   JCPU   PCPU   WHAT
bxg1     tty2     -        01:35    6.00s  0.02s  0.02s  -bash
bxg2     tty3     -        01:35    2.00s  0.02s  0.02s  -bash
itheima  tty1     :0       01:31    5:18   7.19s  0.33s  pam: gdm-password
itheima  pts/0    :0.0     01:31    0.00s  0.34s  2.39s  /usr/bin/gnome-terminal
```

从上述结果可以看出,当前登录用户计数为 4(同一个用户登录两个终端记为两个),其中 tty1 的用户 itheima 是在开机后登录的,pts/0 的 itheima 是 GNOME 虚拟终端中的,tty2 和 tty3 中的 bxg1、bxg2 是另外 2 个虚拟控制台中登录的。FROM 指远程登录的主机名,

LOGIN@指登录时间，IDLE 指用户空闲时间，JCPU 指终端连接的进程占用的 CPU 时间，PCPU 指 WHAT(当前运行进程)占用的 CPU 时间。

另外，当使用 su 命令临时切换用户时，w 命令仍然只显示登录时的用户。这是因为 su 命令本质上并不是重新登录另一个用户，其切换后消耗的资源是计算在登录时的用户身上的。

2.2 正则表达式

正则表达式是学习 Nginx 必备的基础知识。在 Nginx 中，许多功能如重定向、URL 重写、缓存和过期时间配置等功能，都支持使用正则表达式来定义复杂的处理规则。

2.2.1 正则表达式概述

正则表达式(Regular Expression，regexp)是一种描述字符串特征的语法规则，用于验证各种字符串是否匹配(Match)这个特征，进而实现高级的文本查找、替换、截取内容等操作。例如，要在大量的文本中找出符合某个特征的字符串，就将这个特征按照正则表达式的语法写出来，形成一个用于计算机程序识别的模式(Pattern)，然后计算机程序就会根据这个模式到文本中进行匹配。

正则表达式的形成与发展有着悠久的历史，在各种计算机软件中都有广泛应用。例如，在操作系统(UNIX、Linux 等)、编程语言(C、C++、Java、PHP、Python、JavaScript 等)、服务器软件(Apache、Nginx)的使用中都会遇到正则表达式。图 2-6 描述了正则表达式的发展历程。

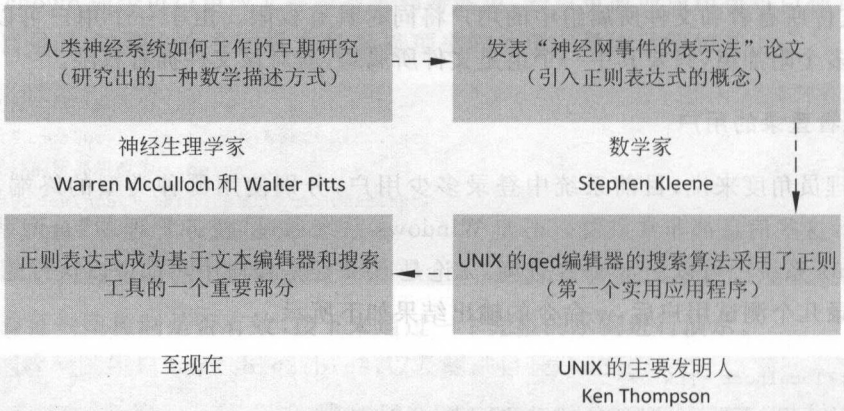


图 2-6 正则表达式的发展

正则表达式在发展过程中出现了多种形式，一种是 POSIX 规范兼容的正则表达式，包括基本语法 BRE(Base Regular Expression)和扩展语法 ERE(Extended Regular Expression)两种规则，用于确保操作系统之间的可移植性，但最终没有成为标准只能作为一个参考。另一种是当 Perl(一种功能丰富的编程语言)发展起来后，衍生出来了 PCRE(Perl Compatible Regular Expressions，Perl 兼容正则表达式)库，使得许多开发人员可以将 PCRE 整合到自己的语言中，Nginx 的许多功能也使用了 PCRE 库。

2.2.2 正则表达式入门

1. grep 命令

在 Linux 中使用正则表达式，可以用 grep(globally search a regular expression and print，以正则表达式全局搜索和打印)命令。grep 最初是 ED 编辑器中的一条命令，后来发展成一个独立的工具。

grep 支持对来自标准输入、管道(pipe)输入，以及文本文件的内容进行正则表达式搜索。接下来分别介绍各种方式下 grep 的使用。

1) 标准输入方式

标准输入就是指通过键盘输入待匹配的内容，具体示例如下。

```
[itheima@localhost ~]$grep --color 'hei'
```

上述示例中，grep 命令的--color 选项表示以彩色标注出匹配到的内容，参数 hei 是一个符合正则表达式语法的匹配模式，用于匹配内容。

该行命令执行后，在光标显示处利用键盘输入 it hei ma 或其他任意内容后，按回车键提交给 grep 进行匹配，结果如下所示。

```
it hei ma
it hei ma
```

上述第 1 行是用户从键盘输入的内容；第 2 行是 grep 输出的匹配结果，其中 hei 显示为红色(本书使用加粗字体表示)。如果没有匹配成功，则没有输出结果。

2) 管道输入方式

管道是 Linux 中支持的一种通信机制，其作用是将一个程序的输出作为另一个程序的输入。管道的符号是|，通过这个符号连接前后多个命令，具体示例如下。

```
[itheima@localhost ~]$ls | grep --color 'Do'
Documents
Downloads
```

上述示例中，ls 命令用于列出当前目录下的文件列表，加上管道符号以后，将输出结果传递给 grep。grep 的参数 Do 匹配出了输出内容中含有 Do 的 2 行结果。另外，使用 ls | cat 命令可以查看 grep 执行前的文本，格式为 1 个文件名占 1 行。

3) 文件方式

grep 命令的第 2 个参数是可选参数，用于读取指定的文件内容进行正则模式匹配。以读取指定文件/etc/passwd 为例，具体示例如下。

```
[itheima@localhost ~]$grep --color 'itheima' /etc/passwd
itheima:x:500:500:itheima.com:/home/itheima:/bin/bash
```

上述示例从/etc/passwd 文件中匹配出了包含 itheima 字符串的一行内容。

2. POSIX 和 Perl 语法

grep 命令支持 POSIX 和 Perl 两种正则表达式语法，默认情况下是 POSIX BRE 语法，指定选项-E 可以切换到 POSIX ERE 语法，指定选项-P 可以切换到 Perl 语法(选项 E 和 P 不可同时使用)。

由于 Nginx 使用了 PCRE，因此下面将基于 Perl 语法进行讲解。下面的示例演示了使用 Perl 语法的正则表达式，找出“以字符 M 开始、字符 c 结束、中间包含任意个字符”的文件名。

```
[itheima@localhost ~]$ls | grep -P '^M.*c$'
Music
```

在上述示例的正则表达式中，“^”匹配一行的开始，“$”匹配一行的结束，“.”用于匹配一个任意字符，“*”表示匹配它前面的字符零次或多次，两者组合起来后可以匹配多个任意字符。

3. 元字符、文本字符和转义字符

一个完整的正则表达式由元字符和文本字符两部分构成。其中，元字符就是具有特殊含义的字符，如前面提到的“^”“$”“.”“*”，文本字符就是普通的文本，如字母和数字等。正则表达式定义许多元字符用于实现复杂匹配，而若要匹配的内容是这些字符本身时，就需要在前面加上转义字符“\”，如“\^”。“\”本身也属于元字符，用“\\”转义。下面的示例演示了转义字符的使用。

```
[itheima@localhost ~]$grep -P --color '[\^\$\*\\]'
123*45$6\78^90
123*45$6\78^90
```

在上述示例中，正则表达式前后的中括号“[ ]”用于匹配里面的字符，只要其中一个字符在输入的字符串中有就会被匹配出来。要匹配的有“^”“$”“*”“\”4 个字符，每个字符前面都加上了转义字符“\”防止被当成元字符识别。从执行结果中可以看出，输入字符串中的特殊字符都被匹配出来。

在书写正则表达式时，使用单引号定界符可以防止空格被 shell 当成参数分隔符的问题，但 shell 的单引号定界符中无法出现单引号，若使用双引号定界符又会出现双重转义的问题。

为了解决这个问题，可以在单引号定界符中用\x27 表示单引号，其中\x 用于将后面的数字 27 当成十六进制数的 ASCII 码所代表的字符。下面的示例演示了匹配字符中含有单引号的情况。

```
[itheima@localhost ~]$grep -P --color '3\x274';
123'456
123'456
```

从执行结果可以看出，正则表达式将输入的字符 3、'、4 匹配出来。

4. 分组

在正则表达式中还支持分组(又称为子模式、子匹配),用小括号“()”来实现。括号用于嵌套一个子模式,如下面的示例实现了匹配 root- 出现了 3 次的情况。

```
[itheima@localhost ~]$grep -P --color '(root-){3}'
root-root-root-root-
root-root-root-root-
```

在上述示例中,{3}用于匹配前面的字符 3 次,如 a{3}表示匹配 aaa。若要匹配的不是一个字符而是一个字符串时,就用括号包裹起来即可。

另外,对于括号中的子表达式的匹配结果,可以用“\数字”来引用,数字是指第几个括号,如第 1 个括号的匹配结果就是“\1”,最大支持 9 个,具体示例如下。

```
[itheima@localhost ~]$grep -P --color '(aa)(bb)\1 \2'
aa bb aa bb aa bb
aa bb aa bb aa bb
```

从上述示例可以看出,grep 命令成功的匹配出前面两组 aa 和 bb。

2.2.3 正则表达式语法规则

前面介绍如何使用 grep 命令实现正则表达式匹配,关于正则表达式的详细语法规则还有很多,下面进行详细讲解。

1. 定位符

在程序开发中,经常需要确定字符的具体位置,如在字符串的头部或尾部。利用正则表达式元字符中的定位符可以实现字符定位,语法说明和示例如表 2-13 所示。

表 2-13 定位符

定位符	说　明	示例	匹配结果
^	匹配字符串开始的位置	^Hello	**Hello** World
$	匹配字符串结尾的位置	World $	Hello **World**

需要注意的是,在使用 grep 命令进行匹配时,只能按行依次匹配,不能实现跨换行符的多行匹配。另外,当需要匹配空行时,可以用^ $ 表示。

2. 选择符

若要查找的条件有多个,只要其中一个满足即可成立时,可以用选择符“|”。该字符可以理解为“或”,具体示例如下。

```
[itheima@localhost ~]$grep -P --color 'Linux|UNIX'
Linux is Good, UNIX is Good!
Linux is Good, UNIX is Good!
```

从以上示例可以看出，字符串中的 Linux、UNIX 都被匹配出来。

3. 字符范围

当需要匹配某个范围内的字符时，可以用中括号“[ ]”和连字符“-”来实现。而且，在中括号中还可以用反义字符“^”，表示匹配不在指定字符范围内的字符。具体说明和示例如表 2-14 所示。

表 2-14 字符范围

示例	说 明	匹配结果
[abc]	匹配字符 a、b、c	**a**pple，ch**e**rry，**b**an**a**na
[^abc]	匹配除 a、b、c 以外的字符	**a**pple，ch**e**rry，b**a**n**a**na
[a-z]	匹配字母 a～z 范围内的字符	1**a**2A34**xyz**56

表 2-14 列举了几个典型的字符范围。除了这些以外，还可以写成其他多种形式，如[^a-z]匹配除字母 a～z 范围外的字符、[a-zA-Z0-9] 匹配字母(包括大写和小写)和数字 0～9 范围内的字符。

需要注意的是，字符“-”在通常情况下只表示一个普通字符，只有在表示字符范围时才作为元字符来使用。“-”连字符表示的范围遵循字符编码的顺序，如 a-Z、z-a、a-9 都是不合法的范围。

4. 点字符和限定符

点字符“.”用于匹配一个任意字符，限定符(?、+、*、{ })用于匹配某个字符连续出现的次数。关于点字符和限定符的详细说明如表 2-15 所示。

表 2-15 点字符和限定符

字符	说 明	示 例	结 果
.	匹配一个任意字符	s.t	可匹配 sat、set、sit 等
?	匹配前面的字符零次或一次	colou?r	可匹配 colour 和 color
+	匹配前面的字符一次或多次	go+gle	可匹配范围从 gogle 到 goo…gle
*	匹配前面的字符零次或多次	go*gle	可匹配范围从 ggle 到 goo…gle
{n}	匹配前面的字符 *n* 次	go{2}gle	只能匹配 google
{n,}	匹配前面的字符最少 *n* 次	go{2,}gle	可匹配范围从 google 到 goo…gle
{n,m}	匹配前面的字符最少 *n* 次，最多 *m* 次	employe{0,2}	可匹配 employ、employe 和 employee 3 种情况

当点字符和限定符连用时，可以实现匹配指定数量范围的任意字符。例如，^start.*end$ 可以匹配从 start 开始到 end 结束，中间包含零个或多个任意字符的字符串。

正则表达式支持贪婪匹配和惰性匹配两种方式，贪婪表示匹配尽可能多的字符，惰性表示匹配尽可能少的字符。在默认情况下是贪婪匹配，若实现惰性匹配时，在上一个限定符的

后面加上“?”符号。具体示例如表 2-16 所示。

表 2-16 贪婪和惰性匹配

匹配方式	示例	说 明	匹配结果
贪婪	a.*b	最先出现的 a 到最后出现的 b	**a00b00bab**c
惰性	a.*?b	最先出现的 a 到最先出现的 b	**a00b**00**ab**c

5. 小括号

小括号有两个作用，一是改变作用范围，二是分组。具体如表 2-17 和表 2-18 所示。

表 2-17 改变作用范围

示 例	说 明	匹配结果
thir|fourth	改变作用范围前	**thir**th **fourth**
(thir|four)th	改变作用范围后	**thirth fourth**

表 2-18 分组

示例	说 明	匹配结果
app{2}	分组前	**appp**pappapp
(app){2}	分组后	appppp**appapp**

在表 2-17 中，小括号实现了匹配 thirth 和 fourth，而如果不使用小括号，则变成 thir 和 fourth。在表 2-18 中，未分组时，表示匹配 2 个 p 字符，而分组后，表示匹配 2 个 app 字符串。

6. 反斜线

反斜线“\”有两个作用，一是作为转义字符；二是表示一些不可打印的字符、指定预定义字符集等。关于转义字符在前面已经讲过，反斜线的一些常用功能如表 2-19 所示。

表 2-19 反斜线的常用功能

字符	说 明
\d	任意一个十进制数字，相当于[0-9]
\D	任意一个非十进制数字
\w	任意一个单词字符，相当于[a-zA-Z0-9_]
\W	任意一个非单词字符
\s	任意一个空白字符(如空格、水平制表符等)
\S	任意一个非空白字符
\b	单词分界符，如\bapple 可以匹配 test **apple**
\B	非单词分界符，如\Bple 可以匹配 test ap**ple**
\xhh	表示 hh(十六进制 2 位数字)对应的 ASCII 字符，如\x61 表示 a

2.2.4 正则表达式应用案例

在了解正则表达式的基本语法后，还需要大量的综合练习，才能够灵活、熟练地运用。下面通过几个应用案例来帮助读者进一步学习、理解和应用正则表达式。

1. 验证文件扩展名

文件扩展名是指文件名末尾“.”以后的部分，如网页文件 html、PHP 文件 php、图像文件 jpg、CSS 样式文件 css。假设给定的是一个文件路径，只允许访问 html、css 和 jpg 扩展名的文件，则正则表达式如下。

```
^.*?\.(html|css|jpg)$
```

在通过 grep 进行验证时，可以添加选项-i 忽略大小写，从而支持 jpg、Html 等形式的文件扩展名。

2. 验证 IP 地址

以 IPv4 地址的格式为例，正确的 IP 地址范围是 0.0.0.0 到 255.255.255.255。下面分步骤讲解如何用正则表达式进行验证。

1）验证 0～255 之间的数字

在验证数字范围时，先将整个范围拆分成 0～99、100～199、200～255 三个小范围，然后用“|”连接起来形成整个范围。具体正则表达式如下。

```
  0~99: [1-9]?\d
100~199: 1\d{2}
200~255: 2([0-4]\d|5[0-5])
  0~255: [1-9]?\d|1\d{2}|2([0-4]\d|5[0-5])
```

2）验证分隔符

IP 地址的分隔符是“.”，其规律是在前 3 个数字的末尾出现，最后一个数字的后面没有。假设验证的 IP 地址为 0.0.0.0，则正则表达式为^(0\.){3}0$。若要验证范围是 0～255，则将 0 改为对应的正则表达式即可。因此，验证 IP 地址的完整正则表达式如下。

```
^(([1-9]?\d|1\d{2}|2([0-4]\d|5[0-5]))\.){3}([1-9]?\d|1\d{2}|2([0-4]\d|5[0-5]))$
```

3. 验证日期格式

日期的书写格式有很多种，这里以常见的“年-月-日”格式为例，如 2016-10-5。其中年份可以是 1000～9999，月份为 1～12，天数为 1～31。不考虑较复杂的不同月份天数不同的问题。实现日期格式验证的正则表达式如下。

```
验证年份:[1-9]\d{3}
验证月份:[1-9]|1[0-2]
验证天数:[1-9]|[1-2]\d|3[01]
完整正则表达式:^[1-9]\d{3}-([1-9]|1[0-2])-([1-9]|[1-2]\d|3[01])$
```

从上述示例可以看出，只要分析出指定格式的规律，就可以通过正则表达式实现验证。

2.3 HTTP 协议

Nginx 是一个 Web 服务器软件，而 HTTP 协议是服务器与浏览器之间重要的交互规范。在 Nginx 中许多功能模块和指令都与 HTTP 协议相关，只有熟悉 HTTP 协议中的一些基础知识后，才能理解 Nginx 的各种功能的具体作用。

2.3.1 HTTP 概述

HTTP(HyperText Transfer Protocol，超文本传输协议)是浏览器与 Web 服务器之间数据交互需要遵循的一种规范。它是由 W3C 组织推出的，专门用于定义浏览器与 Web 服务器之间数据交换的格式。其交互过程如图 2-7 所示。

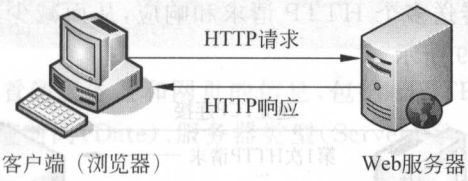


图 2-7 浏览器与 Web 服务器交互过程

HTTP 协议自诞生以来，先后经历了很多版本。目前互联网中应用最多的是 HTTP 1.0 和 HTTP 1.1 版本，下面将针对这两种版本进行详细讲解。

1. HTTP 1.0

基于 HTTP 1.0 协议的客户端与服务器在交互过程中需要经过建立连接、发送请求信息、回送响应信息、关闭连接 4 个步骤，如图 2-8 所示。

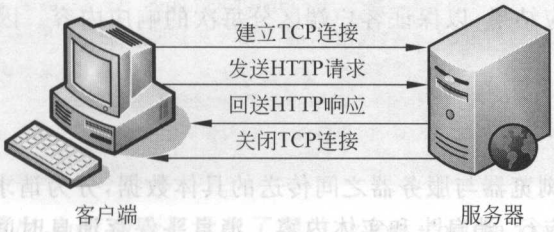


图 2-8 HTTP 1.0 交互过程

在交互步骤中，建立和关闭连接基于 TCP(传输控制协议)，由操作系统实现(如 Linux 的 Socket 编程)，而发送请求是由客户端软件(如浏览器)按照 HTTP 规定的格式向服务器发送请求消息，由服务器端软件(如 Nginx)收到后按照 HTTP 协议解析出具体的内容进行处理。当 Web 服务器处理完成后，为了告知浏览器处理结果，再按照 HTTP 协议回送响应消息，从而完成交互。

HTTP 1.0 方式每次建立 TCP 连接后，只能处理一个 HTTP 请求，这种通信方式对于内容越来越丰富的网页来说，显然效率低下。以下面一段 HTML 代码为例。

```
<html>
    <body>
        <img src="/image01.jpg">
        <img src="/image02.jpg">
        <img src="/image03.jpg">
    </body>
</html>
```

上述HTML文档中包含3个<img>标记，src属性指定了图片的来源是本站目录下的图片地址。当浏览器访问这个网页时，除了网页本身建立了1次连接，这3张图片还要再建立3次连接。如此一来，必然会导致客户端与服务器端交互的耗时，影响网页访问速度。

2. HTTP 1.1

为了克服HTTP 1.0的缺陷，HTTP 1.1版本应运而生。HTTP 1.1支持持久连接，能够在一个TCP连接上传送多个HTTP请求和响应，从而减少建立和关闭连接的消耗和延时。其交互过程如图2-9所示。

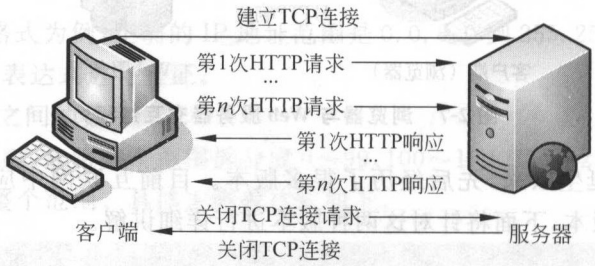


图2-9 HTTP 1.1交互过程

从图2-9中可以看出，当客户端与服务器建立连接后，客户端可以发送多个HTTP请求，并且在发送下一个请求时无须等待上次请求的返回结果。服务器必须按照接收请求的先后顺序依次返回响应结果，以保证客户端区分每次的响应内容。因此HTTP 1.1有效提高了交互效率。

2.3.2 HTTP消息

HTTP消息是指浏览器与服务器之间传送的具体数据，分为请求和响应。一个完整的消息包含请求行或响应行、消息头和实体内容。消息头保存消息时间、系统环境、内容大小和编码格式等信息，实体内容则保存网页或数据。例如，在使用浏览器访问http://www.itheima.com时，浏览器会向域名为www.itheima.com的服务器发送请求消息，请求消息头中包含客户端信息(如操作系统和浏览器的版本、Cookie等)。服务器接到请求后，将网站首页的HTML文档放在响应消息的实体内容中，并通过响应消息头告知服务器端的信息(如服务器的版本、内容的格式等)。

在使用浏览器时，HTTP的消息头是被隐藏的，普通用户只能看到HTML文档渲染后的网页。为了更好地学习HTTP协议，接下来介绍两种查看HTTP消息的方法。

1. curl 查看 HTTP 消息

在 Linux 中，使用 curl 命令可以发送请求并将服务器的响应消息直接显示出来，具体示例如下。

1）发送请求并显示响应消息头

curl 命令的选项-I 用于只显示响应消息头，如果省略该参数则显示实体内容。

```
[itheima@localhost ~]$curl -I http://www.itheima.com
HTTP/1.1 200 OK
Date: Tue, 25 Oct 2016 10:13:59 GMT
Server: Apache/2.2.25(Win32)mod_fcgid/2.3.6
Vary: Accept-Encoding,Cookie
Cache-Control: max-age=3, must-revalidate
Content-Length: 161292
Set-Cookie: expires=Tue, 25-Oct-2016 15:59:59 GMT; domain=itheima.com; path=/
Content-Type: text/html; charset=UTF-8
```

从上述输出结果可以看到服务器和网页的信息，包括 HTTP 协议版本（HTTP/1.1）、响应状态码（200 OK）、响应时间（Date）、服务器类型（Server）等，这些信息的详细解释会在后面提供。

2）发送请求并显示响应的实体内容

下面通过 curl 命令查看响应消息中的实体内容，由于输出结果过长，所以使用 less 进行查看。

```
[itheima@localhost ~]$curl http://www.itheima.com | less
<!DOCTYPE html>
<html lang="zh-CN">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
        <title>黑马程序员官网 | Android 培训|Java 培训|JavaEE 培训|iOS 培训|UI 设计培训|云计算|
        PHP|前端移动开发</title>
        <meta name="keywords" content="黑马程序员 Android 培训,Java 培训 ,JavaEE 培训,iOS 培
        训,UI 设计培训,PHP 培训,云计算培训,Web 前端培训,移动端 WEB 前端培训" />
⋮
```

从上述结果可以看出，curl 命令显示出了网站 www.itheima.com 首页的 HTML 文档内容。

2. 浏览器查看 HTTP 消息

在目前许多主流的浏览器中（如火狐、Google Chrome），已经加入了开发者工具的功能，可以很方便地查看浏览一个网页时发送的所有请求，包括请求消息和响应消息等。以 Chrome 浏览器为例，在浏览器窗口中按 F12 键可以启动开发者工具，然后执行 Network→Headers，如图 2-10 所示。

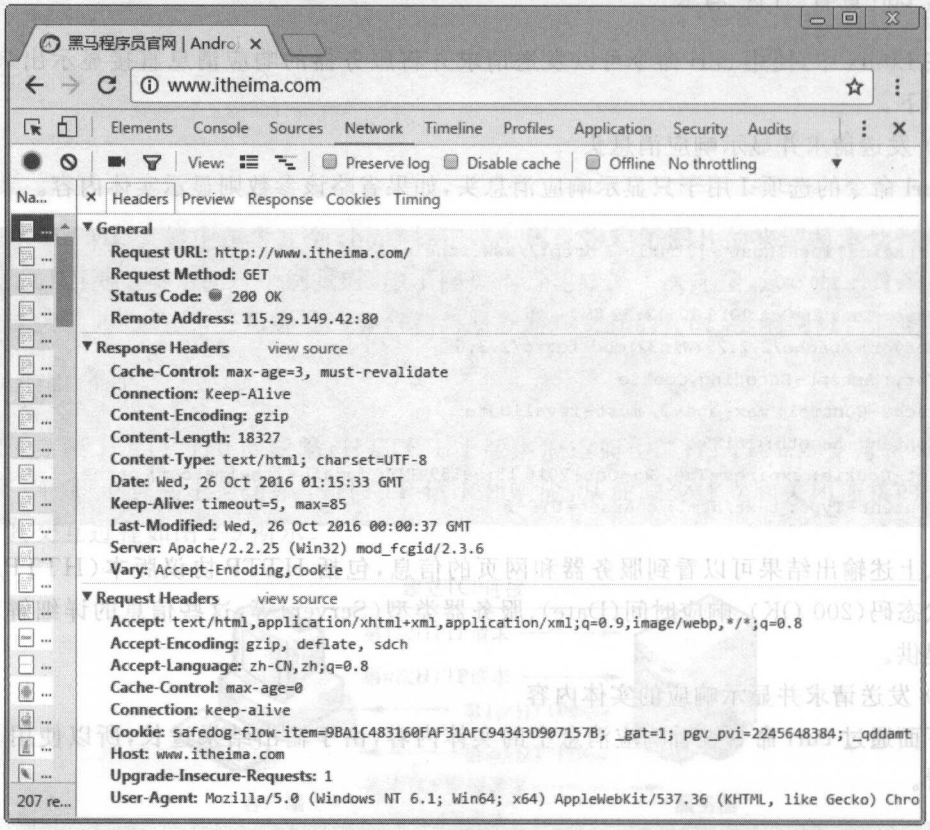


图 2-10 查看 HTTP 消息

2.3.3 HTTP 请求消息

1. 请求行

HTTP 请求消息由请求行、请求头和实体内容三部分组成，请求行位于请求消息的第一行。下面通过示例演示一个典型的请求行。

```
GET /index.php HTTP/1.1
```

上述示例中的请求行共分为三部分，分别是请求方式(GET)、请求资源路径(/index.php)和 HTTP 协议版本(HTTP/1.1)。其中，请求方式有许多种，GET 方式是浏览器打开网页默认使用的方式；请求资源路径是指当访问“http://域名/index.php”URL 地址时，域名后面的部分。

HTTP 协议的多种请求方式具体如表 2-20 所示。常用的请求方式有 GET 和 POST，其中 POST 方式经常用在网页的<form>表单中，在提交表单时，浏览器将用户填写的信息放在请求消息的实体内容发送。

表 2-20　HTTP 请求方式

请求方式	含　义
GET	获取“请求资源路径”对应的资源
POST	向“请求资源路径”提交数据，请求服务器进行处理
HEAD	获取“请求资源路径”的响应消息头
PUT	向服务器提交数据，存储到“请求资源路径”的位置
DELETE	请求服务器删除“请求资源路径”的资源
TRACE	请求服务器回送收到的请求信息，主要用于测试或诊断
CONNECT	保留将来使用
OPTIONS	请求查询服务器的性能，或者查询与资源相关的选项和需求

2. 请求头

请求头位于请求行之后，主要用于向服务器传递附加消息。例如，浏览器可以接受的数据类型、压缩方法、语言以及系统环境。具体示例如下。

```
Host: www.itheima.com
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: safedog-flow-item=9BA1C483160FAF31AFC94343D907157B; _gat=1
```

从上述示例可以看出，每个请求头都是由头字段名称和对应的值构成的，中间用冒号“:”和空格分隔。这些头字段大部分是 HTTP 协议规定的，每个都有特定的用途，而应用程序也可以添加自定义字段。常见的请求头字段和说明如表 2-21 所示。

表 2-21　常见 HTTP 请求头

请　求　头	含　义
Accept	客户端浏览器支持的数据类型
Accept-Charset	客户端浏览器采用的编码
Accept-Encoding	客户端浏览器支持的数据压缩格式
Accept-Language	客户端浏览器所支持的语言包，可以指定多个
Host	客户端浏览器想要访问的服务器主机
If-Modified-Since	客户端浏览器对资源的最后缓存时间
Referer	客户端浏览器是从哪个页面过来的
User-Agent	客户端的系统信息，包括使用的操作系统、浏览器版本号等

续表

请 求 头	含 义
Cookie	客户端需要带给服务器的数据
Cache-Control	客户端浏览器的缓存控制
Connection	请求完成后，客户端希望是保持连接还是关闭连接

3. 提交数据

当利用POST方式提交数据时，数据将被放入实体内容中发送。GET方式没有实体内容，但可以利用URL传递数据。

1) GET

当浏览器通过GET方式提交数据时，数据内容会在URL地址中显示，并通过“?”区分资源路径和提交的数据，这种提交方式也称为URL参数传递，具体示例如下。

```
http://www.itheima.com/test.php?name=xiaoming&password=123456
```

在上述示例中，URL参数由参数名和参数值组成，中间用等号＝连接，多个参数之间用&分隔。因此，上述示例共传递了name和password两个参数，对应的参数值为xiaoming和123456。当浏览器发送请求时，请求行的资源路径为/test.php?name＝xiaoming&password＝123456。

需要注意的是，URL参数遵循URL编码规则，一些特殊符号和中文是无法直接书写的，需要经过编码处理后才能正确使用。目前主流浏览器都支持对地址栏的URL自动编解码，从而使用户体验更加友好，但实际传输时一定是编码后的结果。

2) POST

当<form>表单通过POST方式提交数据时，表单数据有多种编码格式。一个普通的表单示例如下。

```
<form method="post" action="/test.php">
    <input type="text" name="name" value="xiaoming">
    <input type="password" name="password" value="123456">
    <input type="submit">
</form>
```

上述表单中有文本框、密码框和提交按钮，表单的提交方式(method属性)为post，提交的目标地址为test.php。当表单提交后，发送的HTTP请求消息头如下。

```
POST /test.php HTTP/1.1
Host: www.itheima.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

name=xiaoming&password=123456
```

从上述示例可以看出，当使用POST方式提交表单时，Content-Type消息头字段会自

动设置为 application/x-www-form-urlencoded，表示以 URL 编码为格式的表单，Content-Length 消息头会自动设置为实体内容的长度(29 字节)。

表单默认的编码格式不支持上传文件。当需要利用表单上传文件时，需要为表单添加 enctype 属性，将编码格式(Content-Type)设置为 multipart/form-data。这种格式可以将表单数据和上传文件数据一起编码到实体内容中，由服务器端进行读取。

2.3.4　HTTP 响应消息

1. 响应状态行

当服务器收到浏览器的请求后，就会在处理完成后回送响应消息给浏览器。有些浏览器在遇到网速慢或者服务器处理慢时，就会出现“正在等待服务器响应”的提示。

在 HTTP 响应消息中，位于第一行的是状态行，用于告知浏览器本次响应的状态，示例如下。

```
HTTP/1.1 200 OK
```

上述示例中，HTTP/1.1 是协议版本，200 是状态码，OK 是状态的描述信息。

响应状态码表示服务器对客户端请求的各种不同的处理结果和状态，由一个三位十进制数表示。响应状态码共分为 5 个类别，通过最高位的 1～5 来分类。

- 1xx：成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程。
- 2xx：成功接收请求并已完成整个处理过程。
- 3xx：为完成请求，客户端需进一步细化请求。
- 4xx：客户端的请求有错误。
- 5xx：服务器端出现错误。

HTTP 协议定义的状态码非常多，对于普通用户来说无须每个都要深入研究，但对于其中一些经常遇到的状态码要了解，具体如表 2-22 所示。

表 2-22　常见响应状态码

状　态　码	含　义
200(正常)	客户端的请求成功，响应消息返回正常的请求结果
301(永久移动)	被请求的文档已经被移动到别处，此文档的新 URL 地址为响应头 Location 的值，浏览器以后对该文档的访问会自动使用新地址
302(找到)	和 301 类似，但是 Location 返回的是一个临时的、非永久 URL 地址
304(未修改)	浏览器在请求时会通过一些请求头描述该文档的缓存情况，当服务器判断文档没有修改时，就通过 304 告知浏览器继续使用缓存，否则服务器将使用 200 状态码返回修改后的新文档
401(未经授权)	当浏览器试图访问一个受密码保护的页面时，且在请求头中没有 Authorization 传递用户信息，就会返回 401 状态码要求浏览器重新发送带有 Authorization 头的信息
403(禁止)	服务器理解客户端的请求，但是拒绝处理。通常由服务器上文件或目录的权限设置导致

续表

状 态 码	含 义
404(找不到)	服务器上不存在客户端请求的资源
500(内部服务器错误)	服务器内部发生错误,无法处理客户端的请求
502(无效网关)	服务器作为网关或者代理访问上游服务器,但是上游服务器返回了非法响应
504(网关超时)	服务器作为网关或者代理访问上游服务器,但是未能在规定时间内获得上游服务器的响应

2. 响应头

响应头位于响应状态行的后面,用于告知浏览器本次响应的一个基本信息,包括服务程序名、内容的编码格式、缓存控制等。常见的 HTTP 响应头如表 2-23 所示。

表 2-23 常见 HTTP 响应头

响 应 头	含 义
Server	服务器的类型和版本信息
Date	服务器的响应时间
Expires	控制缓存的过期时间
Location	控制浏览器显示哪个页面(重定向到新的 URL)
Accept-Ranges	服务器是否支持分段请求,以及请求范围
Cache-Control	服务器控制浏览器如何进行缓存
Content-Disposition	服务器控制浏览器以下载方式打开文件
Content-Encoding	实体内容的编码格式
Content-Length	实体内容的长度
Content-Language	实体内容的语言和国家名
Content-Type	实体内容的类型和编码类型
Last-Modified	请求文档的最后一次修改时间
Transfer-Encoding	文件传输编码
Set-Cookie	发送 Cookie 相关的信息
Connection	是否需要持久连接

HTTP 协议的请求头和响应头是浏览器与服务器之间交互的重要信息,由浏览器和 Web 服务器自动处理,通常不需要人为干预。但有时 Web 开发者会需要手动更改一些响应消息,以实现网站项目的某些功能需求,或者进行浏览器缓存方面的优化。在学习 Nginx 的过程中,也会遇到诸多与响应头相关的模块和指令,具体会在用到时进行讲解,此处只需简单了解即可。

3. 实体内容

服务器响应的实体内容有多种编码格式。当用户请求的是一个网页时，实体内容的格式就是 HTML。如果请求的是图片，则响应图片的数据内容。服务器为了告知浏览器内容类型，会通过响应消息头中的 Content-Type 来标识。例如，网页的类型通常是“text/html; charset=UTF-8”，表示内容的类型为 HTML，字符集是 UTF-8，其中 text/html 是一种 MIME 类型表示方式。

MIME 是目前在大部分互联网应用程序中通用的一种标准，其表示方法为“大类别/具体类型”。一些常见的 MIME 类型如表 2-24 所示。

表 2-24　常见 MIME 类型

MIME 类型	说　明	MIME 类型	说　明
text/plain	普通文本(.txt)	text/css	CSS 文件(.css)
text/xml	XML 文档(.xml)	application/javascript	JavaScript 文件(.js)
text/html	HTML 文档(.html)	application/x-httpd-php	PHP 文件(.php)
image/gif	GIF 图像(.gif)	application/rtf	RTF 文件(.rtf)
image/png	PNG 图像(.png)	application/pdf	PDF 文件(.pdf)
image/jpeg	JPEG 图像(.jpg)	application/octet-stream	任意的二进制数据

浏览器对于服务器响应的不同 MIME 类型会有不同的处理方式，如遇到普通文本时直接显示，遇到 HTML 时渲染成网页，遇到 GIF、PNG、JPEG 等类型时显示为图像。如果浏览器遇到无法识别的类型，在默认情况下会执行下载文件的操作。

本章小结

本章讲解 Linux 入门、正则表达式和 HTTP 协议 3 方面的基础知识内容。通过本章的学习，读者应该熟练掌握 Linux 的基本操作命令，能够对文件、用户和权限进行管理操作；学会运用正则表达式对一些常见的文本格式进行匹配或验证；学会查看 HTTP 消息，了解一些常见的请求方式、请求头、响应头、状态码和内容编码格式等，为后面的学习奠定基础。

课后练习

一、填空题

1. 一个完整的正则表达式由_____和文本字符两部分构成。
2. Linux 系统中并没有盘符的概念，而是采用一切都是从_____开始。

二、判断题

1. Linux 中的所有命令都可以通过--help 选项查看具体帮助。　(　　)

2. 用户的账号分为多种类型，不同账号拥有的权限、担任的角色也各不相同。（　　）

三、选择题

1. 下列选项中，属于shell脚本文件的扩展名的是(　　)。

A. s　　B. h　　C. sh　　D. c

2. 下列选项中，命令ls -a作用的目录是(　　)。

A. 当前目录　　B. 当前主机登录用户的家目录

C. 超级用户目录　　D. 操作当前主机的用户家目录

3. 当源文件存在，而目标文件不存在时，mv命令执行(　　)操作。

A. 移动　　B. 重命名

C. 移动并重命名　　D. 以上答案都不是

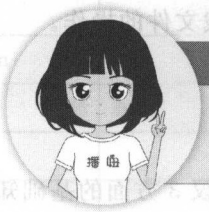
四、简答题

1. 请列举出Linux根目录下常用的文件及其作用。(不少于6个)。

2. 请简述常用的文件查看搜索命令及其功能特点。(至少4种)。

五、操作题

在Linux系统中，硬盘的各个分区是挂载到“/”根目录下的，使用mount命令可以实现分区挂载。请尝试通过VMware在虚拟机中增加一块硬盘，然后在CentOS中对硬盘进行分区、格式化，最后利用mount命令挂载分区到一个自定义目录中，并实现开机自动挂载。



第 3 章

Nginx的安装

学习目标

- 熟悉 Linux 服务器环境的搭建；
- 掌握 Nginx 的安装和基本使用；
- 掌握 Nginx 运行环境的基本部署。

在使用 Nginx 之前，首先需要搭建服务器环境，并安装一些必备的软件。本章将在 Windows 系统中通过 VMware 虚拟机来部署网络环境并搭建 Linux 服务器，同时考虑到 Linux 系统的复杂多样，下面基于 CentOS 最小化安装的方式进行讲解，以便读者更好地学习 Nginx 的安装与使用。

3.1 Linux 服务器搭建

Linux 系统以免费、可靠、安全、稳定等优点，在服务器领域广泛应用。在互联网环境下，服务器要求具有很强的稳定性，能够长时间持续稳定工作，因此搭建一个稳定、可靠、安全的服务器非常重要。本节将针对 Linux 服务器的搭建进行详细讲解。

3.1.1 最小化安装 CentOS

前面已经讲解如何利用 VMware 虚拟机快捷安装 Linux 系统。在实际应用时，对于大部分中小企业而言，服务器的采购和维护成本非常昂贵，因此会由专业的运维人员对服务器进行优化，从而最大化地利用硬件资源。所以 Linux 系统的安装和配置对于搭建服务器而言非常重要。

在安装 CentOS 作为服务器来使用时，推荐读者使用最小化安装的方式进行学习。最小化安装是指在安装系统时只预装最基本的组件，然后由用户自行安装其他软件。通过这种方式，可以尽可能地节省磁盘和内存空间，防止不必要的资源浪费。

1. 创建虚拟机

使用 VMware 虚拟机可以很方便地部署实验环境，用户只需要一台计算机，就能够运行多台虚拟的服务器，并支持多种网络环境。

由于采用最小安装的方式，读者既可以使用前面讲过的 CentOS-6.8-x86_64-bin-DVD1.iso 镜像进行安装，也可以下载迷你版镜像 CentOS-6.8-x86_64-minimal.iso 进行安装。

在使用 VMware 创建虚拟机时，可以对硬件参数进行配置。本书配置的硬件参数如表 3-1 所示。

表 3-1 虚拟机的硬件参数

硬　件	说　明
处理器	请根据物理机的核心数设置虚拟机的总核心数(数量越多处理速度越快)
内存	最小支持 512MB(但不支持图形安装界面)，推荐 1024MB
网络连接	使用 NAT(网络地址转换)
磁盘空间	推荐 10GB 或以上
USB 控制器	不需要此虚拟设备
声卡	不需要此虚拟设备
打印机	不需要此虚拟设备

在表 3-1 中，当内存设置为 512MB 时，CentOS 将不会启动图形化安装界面，并且只能进行最小化安装。接下来将具体的操作步骤进行图文演示，具体步骤如下。

(1) 启动 VMware，执行“文件”→“新建虚拟机”命令。

(2) 在弹出的“新建虚拟机向导”页面中选择“典型(推荐)”，然后单击“下一步”按钮。

(3) 在“安装客户机操作系统”页面，选择“稍后安装操作系统”单选按钮，如图 3-1 所示。选择后单击“下一步”按钮。

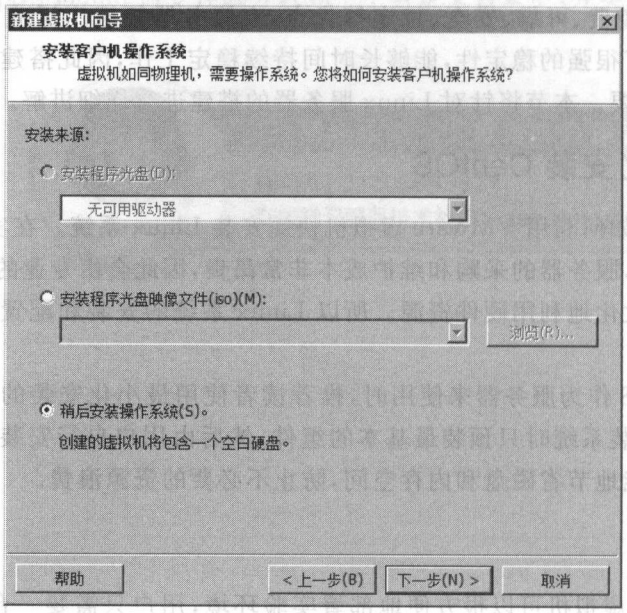


图 3-1 安装客户机操作系统

需要注意的是，如果此处选择“安装程序光盘映像文件(iso)”单选按钮并浏览到 CentOS 的光盘镜像文件，VMware 会自动检测镜像中的操作系统版本并启动快捷安装功能。快捷安装方式虽然可以很轻松地完成系统的安装，但是并不利于学习更深度的手动定

制安装，因此推荐读者选择“稍后安装操作系统”单选按钮。

(4) 在选择客户机操作系统时，选择 Linux 系统，版本选择 CentOS，然后单击“下一步”按钮，如图 3-2 所示。

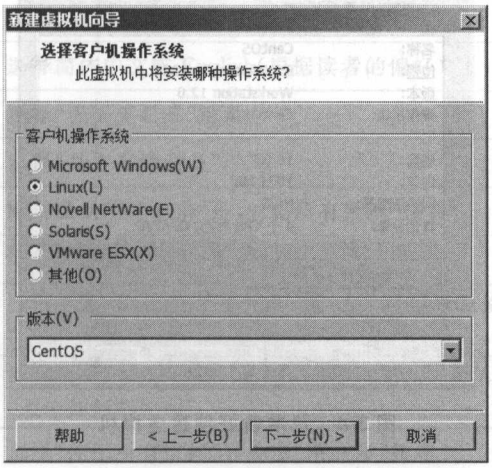


图 3-2 选择操作系统版本

值得一提的是，由于不同的操作系统对于硬件的配置要求不同，因此 VMware 虚拟机对于支持的操作系统提供了推荐的默认配置。当我们正确选择操作系统版本后，VMware 就会根据系统版本来决定硬件的推荐配置。

(5) 配置虚拟机的名称和保存位置，读者可根据物理机的实际情况进行选择，然后单击“下一步”按钮，如图 3-3 所示。

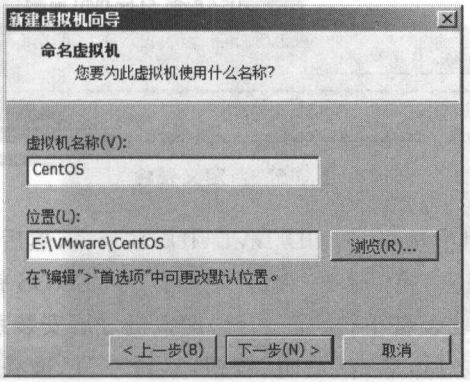


图 3-3 配置虚拟机的名称和保存位置

(6) 指定磁盘容量大小，此处配置 10GB 的空间即可，如果物理机支持大于 4GB 以上的单文件，则选择“将虚拟磁盘存储为单个文件”。在完成此处的配置后单击“下一步”按钮。

(7) 在“已准备好创建虚拟机”页面进行硬件定制，如图 3-4 所示。

(8) 单击“自定义硬件”按钮，弹出如图 3-5 所示的“硬件”对话框。

在图 3-5 中，由于服务器用不到声卡、USB 控制器、打印机这些设备，因此可以将这些设备移除。对于虚拟机的内存和处理器，根据物理机的硬件能力来分配即可。

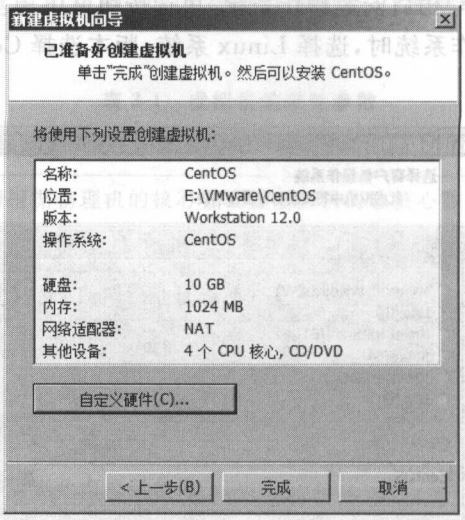


图 3-4 已准备好创建虚拟机

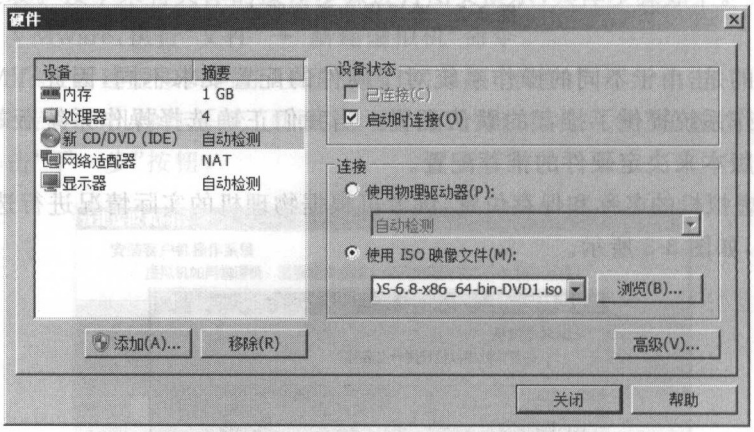


图 3-5 自定义硬件

接着,在左侧设备栏目中选择"新 CD/DVD(IDE)"设备,然后在右侧"连接"分组框中选择"使用 ISO 映像文件"单选按钮,单击"浏览"按钮找到 CentOS 镜像文件,将 CentOS 的光盘镜像装载到虚拟机的光驱设备中,从而在开启虚拟机之后安装系统。

(9) 在硬件定制完成之后,单击"关闭"按钮,然后在"已准备好创建虚拟机"页面单击"完成"按钮即可完成虚拟机的创建。

2. 安装系统

关于 CentOS 的安装步骤,前面是通过 VMware 的快捷安装功能来完成的。而本次在创建虚拟机时并没有开启快捷安装功能,因此在启动虚拟机后会读取光盘镜像进入安装向导。关于最小化安装方式在安装过程中的必要配置如表 3-2 所示。

表 3-2　CentOS 安装说明

安装步骤	配置说明
启动菜单	选择 Install or upgrade on existing system(安装或升级系统)
发现光盘(Disc Found)	选择 Skip 跳过光盘检查
安装语言选择	选择简体中文或 English(根据读者的偏好)
键盘类型选择	选择“美国英语式”键盘
您的安装将使用哪种设备	选择“基本存储设备”
以下设备中可能包含数据	维持默认值并单击“是,忽略所有数据”按钮
请为这台计算机命名	维持默认值并单击“下一步”按钮
请选择离本地时区最近的城市	城市选择“亚洲/上海”,并取消选中“系统时钟使用 UTC 时间”。然后单击“下一步”按钮
请为根用户输入一个密码	即配置 root 用户的密码,可以输入 123456 并单击“下一步”按钮,然后选择“无论如何都使用”。
您要进行哪种类型的安装	选择“使用所有空间”并单击“下一步”按钮
将存储配置写入磁盘	选择“将修改写入磁盘”
选择 CentOS 安装的软件	选择 Minimal(最小安装)并单击“下一步”按钮

在按照表 3-2 中的说明进行操作以后,系统就会开始进行安装,等待安装完成即可。在安装完成后,单击“重新引导”按钮重新启动系统。

在完成安装并重启之后,就会出现 CentOS 的登录界面,如图 3-6 所示。在登录界面中输入用户名 root 和密码 123456 之后就可以登录系统。

图 3-6　CentOS 登录界面

多学一招：查看服务器内存和磁盘空间

在使用最小安装方式安装完成系统后,可以查看当前服务器的内存和磁盘空间,具体方法如下。

(1) 查看内存空间。

使用 free 命令可以查看服务器的内存空间,选项-m 表示以 MB(兆字节)的数据存储单位进行显示。执行结果如下所示:

```
[root@localhost ~]#free -m
             total       used       free     shared    buffers     cached
Mem:           995        168        827          0          6         41
-/+buffers/cache:        121        874
Swap:         1023          0       1023
```

在以上输出结果中，Mem表示系统的物理内存，total表示内存的总大小(995MB)，used表示已经使用的空间(168MB)，free表示可用空间(827MB)。关于其他内容读者可参考Linux帮助手册进行学习，这里就不再详细解释。

(2) 查看磁盘空间。

使用df命令可以查看服务器的磁盘空间，选项-lh表示利用方便阅读的数据存储单位显示本地文件系统。执行结果如下所示：

```
[root@localhost ~]#df -lh
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root
                      8.3G  646M  7.2G   9% /
tmpfs                 498M     0  498M   0% /dev/shm
/dev/sda1             477M   28M  425M   7% /boot
```

在以上输出结果中，Filesystem是文件系统，Size表示该分区的总大小，Used表示已经使用的空间，Avail表示可用空间，Use%表示已经使用的百分比，Mounted on表示挂载路径。可以看出，系统共分为3个文件系统，其中/dev/mapper/VolGroup-lv_root是挂载到根目录的文件系统，总大小为8.3GB，已经使用的空间为646MB。

通过上述命令可以看出，最小安装后的CentOS占用的内存和磁盘空间都非常小。用户可以根据自己的需要来手动安装软件和配置系统，避免因为大量预装软件而导致内存、磁盘等硬件资源的浪费。

3.1.2 网络配置

Linux服务器可以提供的服务包括Web、FTP、DNS、DHCP、数据库和邮箱等多种类型，但是这些服务都离不开网络环境。下面将针对VMware虚拟机和Linux服务器的网络配置进行详细讲解。

1. VMware网络配置

VMware虚拟机提供了虚拟网络功能，可以很方便地进行网络环境部署。在程序的菜单栏中执行“编辑”→“虚拟网络配置”命令，打开如图3-7所示的对话框，查看网络配置。

从图3-7中可以看出，VMware提供了桥接、NAT(网络地址转换)和仅主机模式，分别对应的名称为VMnet0、VMnet8和VMnet1。关于这3种模式的具体介绍如下。

1) 桥接模式

当虚拟机的网络处于桥接模式时，相当于这台虚拟机与物理机同时连接到一个局域网，这两台机器的IP地址将处于同一个网段中。以目前家庭普遍使用的宽带上网环境为例，其网络结构如图3-8所示。

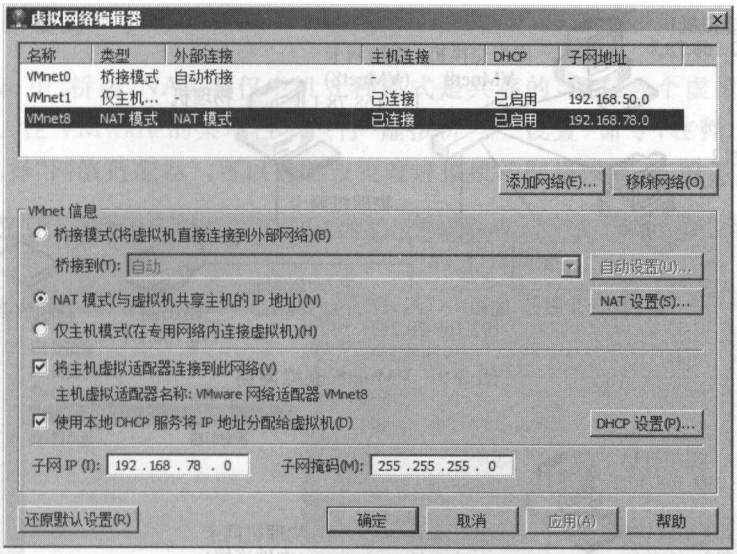


图 3-7　VMware 虚拟网络编辑器

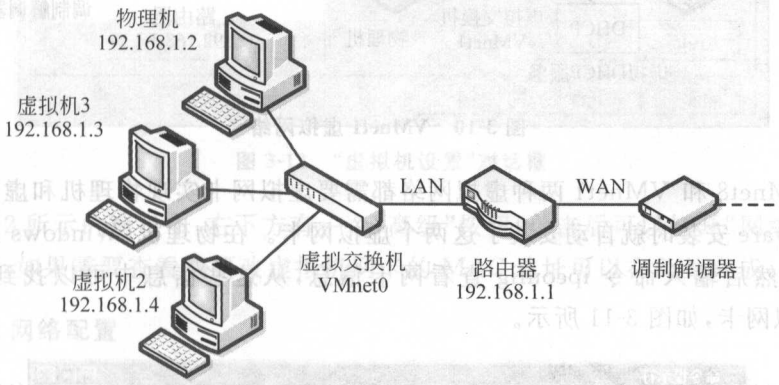


图 3-8　VMnet0 虚拟网络

从图 3-8 可以看出，两台虚拟机和一台物理机同时处于一个局域网内(VMware 支持同时运行多个虚拟机)，如果路由器已经接入网络，则图 3-8 中的 3 台计算机都可以访问外部网络。

2) NAT 模式

NAT 是 VMware 虚拟机中默认使用的模式，其最大的优势是虚拟机接入网络非常简单，只要物理机可以访问网络，虚拟机就可以访问网络。其网络结构如图 3-9 所示。

从图 3-9 可以看出，物理机网卡和 VMnet8 虚拟网络中的 NAT(网络地址转换)网关共享了同一个 IP 地址 192.168.1.2，因此只要物理机连上网，虚拟机就能上网。为了让物理机和虚拟机能够直接互访，需要在物理机中增加一个虚拟网卡接入 VMnet8 虚拟交换机中。

3) 仅主机模式

仅主机模式与 NAT 模式相似，但是在该网络中没有虚拟 NAT，因此只有物理机能上网而虚拟机无法上网，只能在 VMnet1 虚拟网内相互访问。其网络结构如图 3-10 所示。

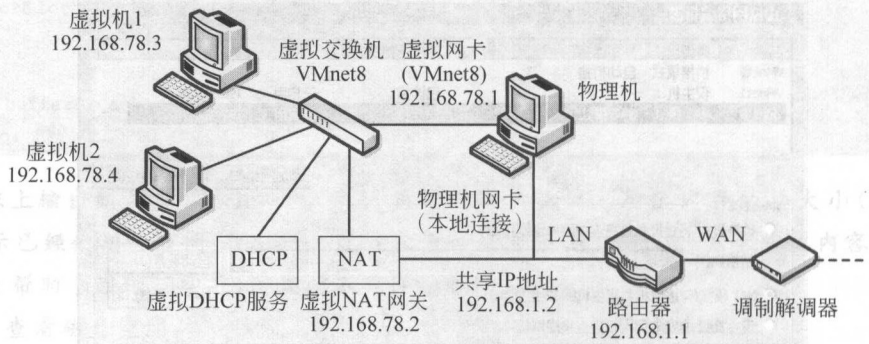


图 3-9 VMnet8 虚拟网络

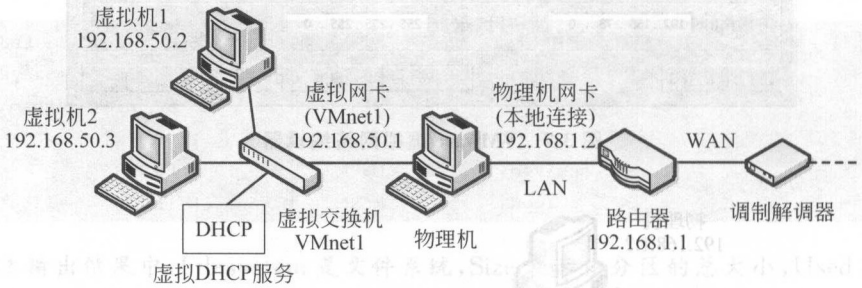


图 3-10 VMnet1 虚拟网络

由于 VMnet8 和 VMnet1 两种虚拟网络都需要虚拟网卡实现物理机和虚拟机的互访，因此在 VMware 安装时就自动安装了这两个虚拟网卡。在物理机(Windows 系统)中打开命令提示符，然后输入命令 ipconfig 查看网卡信息，从这些信息中可以找到 VMent8 和 VMnet1 虚拟网卡，如图 3-11 所示。

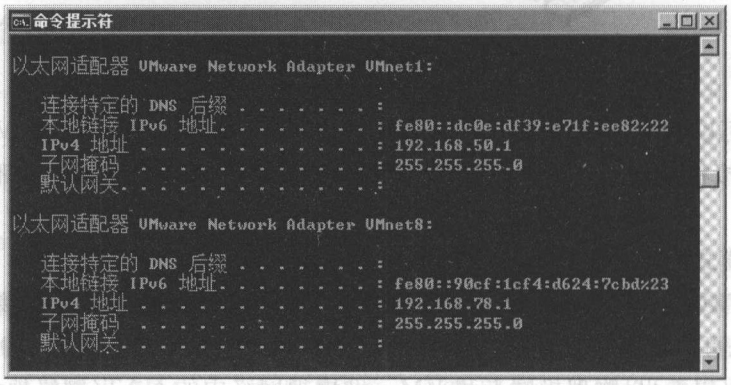


图 3-11 查看 VMware 虚拟网卡

从图 3-11 中可以看出，VMnet1 的 IP 地址为 192.168.50.1，VMnet8 的 IP 地址为 192.168.78.1。这两个 IP 地址是根据 VMware 虚拟网络编辑器中的子网 IP 来自动设置的，如果更改了子网 IP，则这两个网卡的 IP 地址会由 VMware 自动更新。

2. 更改网络模式

在 VMware 中，桥接、NAT 和仅主机 3 种模式是共存的，但是一个虚拟机只能选择一种模式来使用。在 VMware 的菜单栏中执行“虚拟机”→“设置”命令，在弹出的“虚拟机设置”对话框中选择“网络适配器”，可以查看或更改虚拟机的网络模式，如图 3-12 所示。

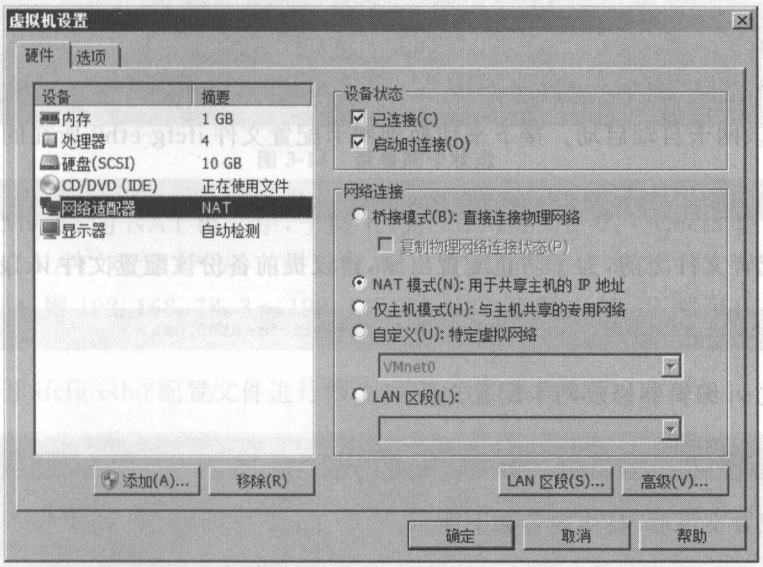


图 3-12　“虚拟机设置”对话框

在图 3-12 所示的窗口中，右下方有一个“高级”按钮，单击后可以打开“网络适配器高级设置”对话框，如果需要查看或更改虚拟机网卡的 MAC 地址可以在此处完成。

3. Linux 网络配置

在了解 VMware 虚拟机的网络环境以后，接下来对 Linux 服务器进行网络配置。在 Linux 系统中，通过 ifconfig -a 命令可以查看所有的网卡，如图 3-13 所示。

```
[root@localhost ~]# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:0C:29:48:2A:8A
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

图 3-13　查看网卡

从图 3-13 中可以看出，目前系统中共有 2 个网卡，第 1 个是 eth0(即编号为 0 的以太网卡)，第 2 个是 lo(即本地回环网卡)。其中 eth0 网卡用于访问外部网络，默认情况下是关闭

的；lo网卡用于在本机内部访问，IP地址为127.0.0.1（即Loopback Address，本机回送地址）。

如果使用VMware的NAT模式或仅主机模式，那么网络中的虚拟机可以通过DHCP（动态主机配置协议）自动获取IP地址。但是在真实环境中，应该为所有的服务器配置静态IP地址，从而确保通过一个IP地址就能找到一台服务器。下面分别介绍如何配置动态和静态IP地址。

1）动态IP

为了使eth0网卡工作，可以通过ifup eth0命令临时启动网卡，也可以修改eth0网卡的配置文件，使该网卡自动启动。接下来切换到网卡配置文件ifcfg-eth0所在的目录：

```
[root@localhost ~]#cd /etc/sysconfig/network-scripts
```

在修改配置文件之前，为了防止配置出错，建议提前备份该配置文件ifcfg-eth0：

```
[root@localhost network-scripts]#cp ifcfg-eth0 ifcfg-eth0.bak
```

然后通过vi编辑器修改网卡配置文件：

```
[root@localhost network-scripts]#vi ifcfg-eth0
```

在打开配置文件后，具体内容如下所示。

```
DEVICE=eth0
HWADDR=00:0C:29:48:2A:8A
TYPE=Ethernet
UUID=de5dcf98-9e30-4d0e-a578-4ddcea528ae6
ONBOOT=no
NM_CONTROLLED=yes
BOOTPROTO=dhcp
```

在上述配置中，需要重点关注的是ONBOOT和BOOTPROTO两个配置，其他配置保持默认值即可。其中BOOTPROTO用于设置获取IP的方式是动态还是静态的，此处默认值为dhcp表示动态获取IP；ONBOOT用于设置网卡是否自动启动，默认值为no，更改为yes即可实现自动启动。

修改完成后，保存并退出编辑，然后执行重新加载网络服务的命令service network reload使配置生效。在配置生效之后，通过ifconfig命令查看eth0网卡的状态，如图3-14所示。

从图3-14中可以看出，eth0网卡已经获取到IP地址192.168.78.128，说明服务器已经成功连接到NAT网络中。如果在重新加载网络服务时报错，则可能是网卡配置文件更改有误，或VMware虚拟网络配置有误，按照前面讲解的内容检查并更正即可。

2）静态IP

静态IP是用户手动设置的IP，设置后固定不变。只要将ifcfg-eth0配置文件中BOOTPROTO的值设置为static，将IPADDR（IP地址）的值设置为其所在子网中正确的、无冲突的IP地址即可。

```
[root@localhost network-scripts]# service network reload
Shutting down interface eth0:                              [  OK  ]
Shutting down loopback interface:                          [  OK  ]
Bringing up loopback interface:                            [  OK  ]
Bringing up interface eth0:
Determining IP information for eth0... done.
                                                           [  OK  ]
[root@localhost network-scripts]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:48:2A:8A
          inet addr:192.168.78.128  Bcast:192.168.78.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe48:2a8a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:182 errors:0 dropped:0 overruns:0 frame:0
          TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17692 (17.2 KiB)  TX bytes:3942 (3.8 KiB)
```

图 3-14　查看网卡状态

假设在 VMware 的 NAT 模式中，子网 IP 为 192.168.78.0、VMnet8 虚拟网卡 IP 为 192.168.78.1、NAT 网关 IP 为 192.168.78.2、DHCP 地址池为 192.168.78.128～192.168.78.254，则 192.168.78.3～192.168.78.127 范围内的 IP 都可以作为静态 IP 使用。

接下来打开 ifcfg-eth0 配置文件进行修改，修改后的示例如下。

```
…(此处省略了前面几行)
BOOTPROTO=static
IPADDR=192.168.78.3
NETMASK=255.255.255.0
GATEWAY=192.168.78.2
DNS1=192.168.78.2
```

上述配置将 BOOTPROTO 的值由 dhcp 修改为 static，然后增加了 IPADDR(IP 地址)、NETMASK(子网掩码)和 GATEWAY(网关)和 DNS1(首选域名服务器)。其中，若网关不设置，虚拟机只能在局域网内访问，无法访问外部网络；若 DNS 不设置，则无法解析域名。

修改配置文件后执行 service network reload 命令使配置生效即可。在配置生效后，可以通过如下操作查看当前使用的默认网关、DNS 服务器。

```
[root@localhost ~]# route | grep default
default         192.168.78.2    0.0.0.0         UG    0      0        0 eth0
[root@localhost ~]# cat /etc/resolv.conf
nameserver 192.168.78.2
```

4. 访问测试

在完成 Linux 服务器的网络配置以后，就可以进行访问测试。无论是 Windows 还是 Linux 系统，都提供了 ping 命令用于检测网络是否连通。在物理机(Windows 系统)中打开命令提示符，执行“ping 虚拟机 IP 地址”，运行结果如图 3-15 所示。

从图 3-15 中可以看出，物理机共向 IP 地址 192.168.78.3 发送了 4 次 ping 请求，4 次都是成功的，发送的数据包为 32 字节，响应时间小于 1 毫秒，TTL(生存时间值)值为 64。其中 TTL 在发送时的默认值为 64，每经过一个路由则减 1，此处显示的最终结果为 64，说明中间没有经过路由。

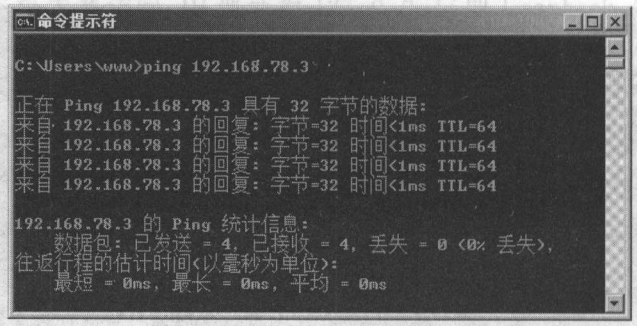


图 3-15 物理机 ping 虚拟机

当使用虚拟机 ping 物理机时，若物理机(Windows)的防火墙为开启的状态，是无法 ping 通的。可以临时关闭 Windows 防火墙，或者将防火墙入站规则中的“文件和打印机共享(回显请求-ICMPv4-In)”设置为“允许连接”。在解决防火墙问题后，虚拟机 ping 物理机的执行结果如图 3-16 所示。

```
[root@localhost network-scripts]# ping 192.168.78.1 -c4
PING 192.168.78.1 (192.168.78.1) 56(84) bytes of data.
64 bytes from 192.168.78.1: icmp_seq=1 ttl=64 time=0.615 ms
64 bytes from 192.168.78.1: icmp_seq=2 ttl=64 time=0.307 ms
64 bytes from 192.168.78.1: icmp_seq=3 ttl=64 time=0.308 ms
64 bytes from 192.168.78.1: icmp_seq=4 ttl=64 time=0.289 ms

--- 192.168.78.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.289/0.379/0.615/0.138 ms
```

图 3-16 虚拟机 ping 物理机

从图 3-16 中可以看出，在 Linux 中使用 ping 命令时加上了参数-c4，表示 ping 的次数为 4。如果省略该参数，则会一直执行，直到按 Ctrl+C 键停止程序。

在测试局域网内的访问后，还需要测试虚拟机能否访问外网。在物理机正确接入外网的前提下，用虚拟机 ping 外部主机(如 ping baidu.com)是可以 ping 通的。如果在正确配置后虚拟机仍然无法访问网络，则有可能是物理机中安装了多个网卡，VMware 会自动使用优先级较高的网卡(无论该网卡是否接入外网)，更改网卡优先级或者禁用这些网卡可以解决问题。在默认情况下，新安装的网卡优先级高于原有网卡，但 VMnet1、VMnet8 两个虚拟网卡的优先级低于本地连接网卡，这样可以避免影响用户正常使用网络。

3.1.3 远程终端访问

当服务器部署好以后，除了直接在服务器上操作，还可以通过网络进行远程连接访问。CentOS 6.8 默认支持 SSH(Secure Shell，安全 Shell 协议)，该协议通过高强度的加密算法提高了数据在网络传输中的安全性，防止中间人攻击(Man-in-the-MiddleAttack，一种黑客常用的攻击手段)。本节将针对如何通过 SSH 远程访问服务器进行详细讲解。

1. SSH 客户端

目前支持 SSH 的客户端有很多，在 Windows 中可以使用 Xshell、SecureCRT 等软件，通过这类软件可以在 Windows 系统上远程控制 Linux 系统。

本书以 Xshell 为例，该软件提供了家庭、学校授权版本，可以免费使用。在 Xshell 的官方网站 http://www.netsarang.com 可以找到软件的下载地址。Xshell 的安装非常简单，按照提示进行操作即可，下面开始分步骤讲解 Xshell 如何使用。

(1) 安装完成以后，打开 Xshell，会自动弹出一个“会话”对话框，如图 3-17 所示。如果关闭了此对话框，也可通过在菜单栏中执行“文件”→“打开”命令再次打开此对话框。

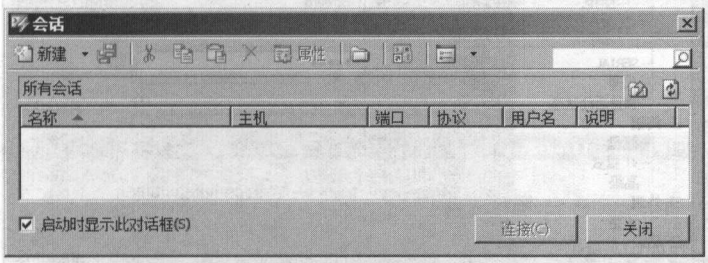


图 3-17　新建会话

(2) 在图 3-17 所示的对话框中，单击工具栏中的“新建”按钮，或在 Xshell 窗口的菜单栏执行“文件”→“新建”命令，会弹出一个“新建会话属性”的对话框，如图 3-18 所示。

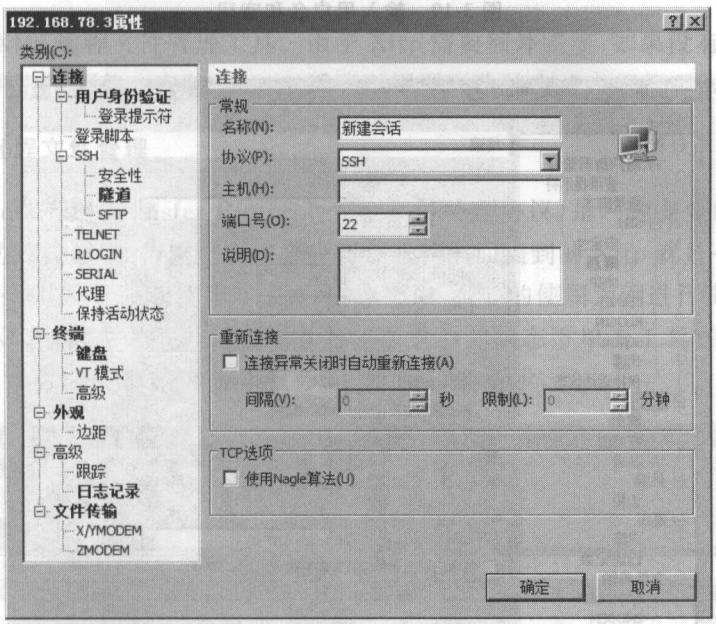


图 3-18　新建会话属性

(3) 在“常规”分组框中输入“名称”和“主机”，其中名称可以随意填写，主机填写服务器的 IP 地址。“协议”选择默认的 SSH 即可，“端口号”保持默认值 22。

(4) 在左侧的“类别”列表中选择“用户身份验证”，然后输入 Linux 服务器的用户名(root)和密码(123456)，如图 3-19 所示。此处输入的用户名和密码会保存到客户端，用于快捷登录。如果考虑安全性，此处可以留空，在每次登录时输入用户名和密码。

(5) 在“类别”列表中选择“终端”，将“终端类型”修改为 linux，如图 3-20 所示。需要注意的是，此处也可以使用默认值 xterm，但键盘中的 Num Lock 数字小键盘区的映射会出现问题。

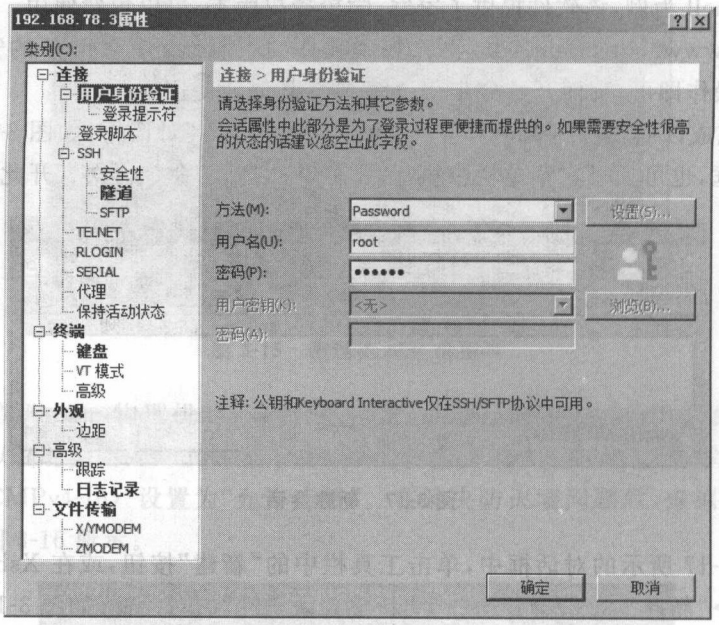


图 3-19 输入用户名和密码

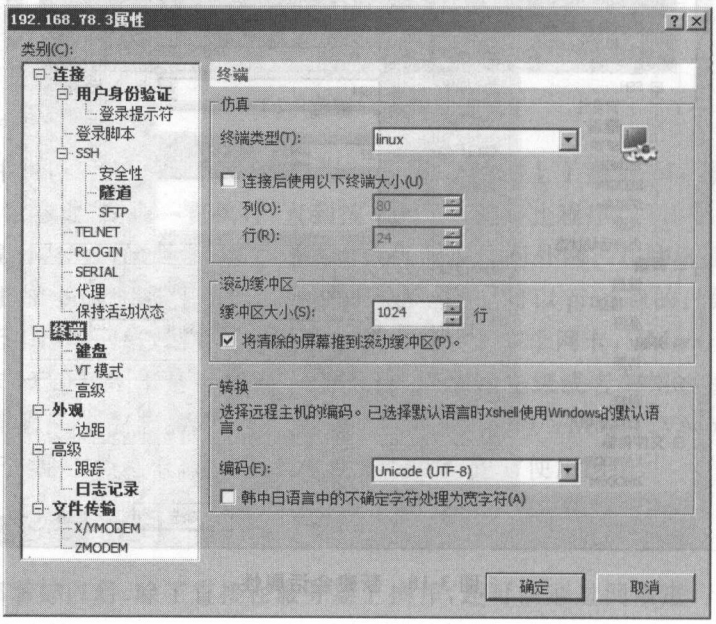


图 3-20 选择终端类型

(6) 设置完成后，单击“确定”按钮保存会话并返回原来的“会话”对话框，如图 3-21 所示。

(7) 选中刚才保存的 192.168.78.3 会话并单击“连接”按钮，即可远程连接到服务器。在连接并登录成功后效果如图 3-22 所示。

值得一提的是，在图 3-22 所示的窗口中，工具栏中有一个“新建文件传输”按钮，通过该

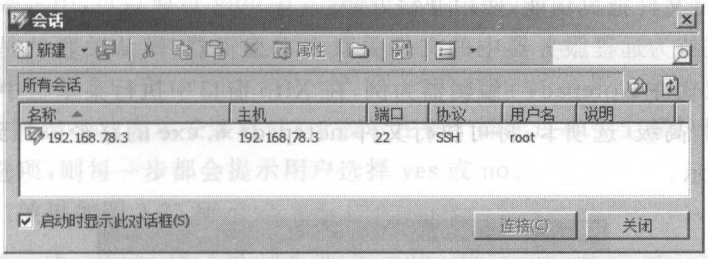


图 3-21　查看保存的会话

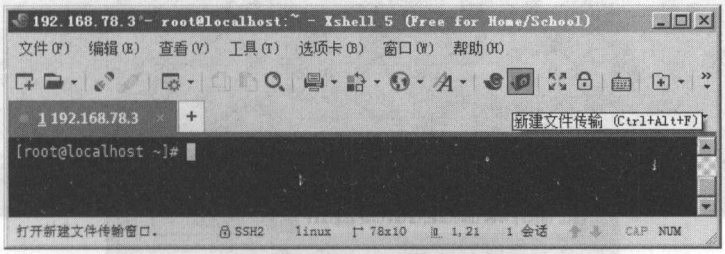


图 3-22　远程登录

按钮可以打开 Xftp 远程文件管理工具。由于 Xftp 需要额外安装，如果没有安装，程序会提示到官方网站中进行下载。安装 Xftp 以后可以用图形化的方式远程管理服务器中的文件。

2. SFTP 远程文件管理

SFTP(Secure File Transfer Protocol，安全文件传送协议)是一种安全的远程文件传输协议，和 SSH 协议类似，在传输过程中会进行加密。前面提到的 Xftp 就是一种 SFTP 的客户端，可以与 Xshell 配合一起使用。接下来，本书以 Xftp 的使用为例进行讲解。

(1) 安装 Xftp 后，在 Xshell 远程服务器登录成功的状态下单击工具栏中的“新建文件传输”按钮可以自动打开 Xftp 并登录服务器，如图 3-23 所示。

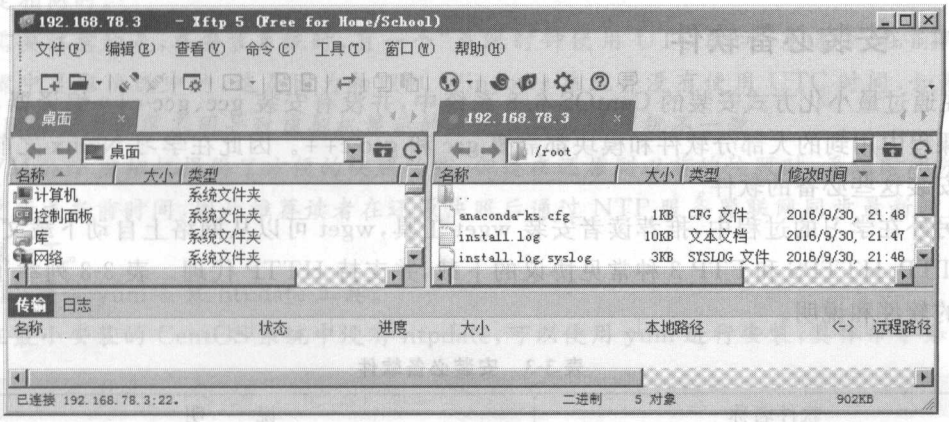


图 3-23　Xftp 远程文件管理

在图 3-23 所示的窗口中，左侧为客户端 Windows 系统的文件列表，右侧为 Linux 系统的文件列表。通过这个窗口，可以实现文件的上传、下载、复制、剪切、删除、修改文件权限和

属性等操作，支持文件拖曳功能，使用非常方便。

(2) Xftp 支持为远程服务器中的文件关联文本编辑器，默认关联的是 Windows 记事本。本书以开源软件 Notepad++编辑器为例，在 Xftp 窗口中执行菜单栏中的“工具”→“选项”操作，切换到“高级”选项卡，将可执行文件 notepad++.exe 的路径添加到“编辑器路径”中，如图 3-24 所示。

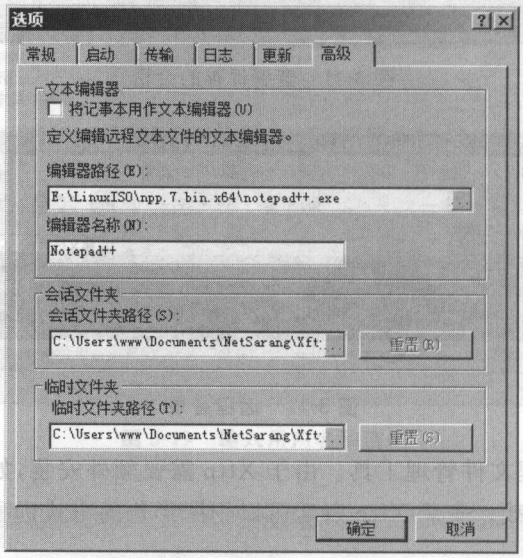


图 3-24 关联文本编辑器

(3) 关联之后，在远程服务器的文件列表中选中一个文件右击，就会出现“以 Notepad++编辑”的一个菜单项，单击后即可调用 Notepad++编辑器自动打开文件。

值得一提的是，在使用 Notepad++创建 Linux 系统中的文件时，推荐将文件的编码格式设置为“UTF8 无 BOM 格式编码”，并且将换行符设置为 UNIX 格式，这样做的目的是保证该文件能够被 Linux 系统中的程序正确识别。

3.1.4 安装必备软件

在通过最小化方式安装的 CentOS 6.8 系统中，并没有安装 gcc、gcc-c++编译器，而后续的课程中用到的大部分软件和模块都需要 gcc 和 gcc-c++。因此在学习 Nginx 之前有必要先安装这些必备的软件。

另外在学习的过程中，推荐读者安装 wget 工具，wget 可以从网络上自动下载文件，支持 HTTP、HTTPS 和 FTP 3 种常见协议的下载，并支持 HTTP 代理。表 3-3 列举了这些必备的软件和说明。

表 3-3 安装必备软件

软件名称	说　明
gcc	用来编译 C 程序
gcc-c++	用来编译 C++ 程序
wget	用来从网络上下载文件

在 CentOS 系统中，可以通过 yum 方式联网自动下载和安装这些软件，具体命令如下：

```
[root@localhost ~]#yum -y install gcc gcc-c++wget
```

在上述命令中，install 表示安装软件；-y 表示在整个操作过程中全部执行默认的 yes 操作，如果省略该选项，则每一步都会提示用户选择 yes 或 no。

安装完成后，效果如图 3-25 所示。

```
Installed:
  gcc.x86_64 0:4.4.7-17.el6        gcc-c++.x86_64 0:4.4.7-17.el6        wget.x86_64 0:1.12-8.el6

Dependency Installed:
  cloog-ppl.x86_64 0:0.15.7-1.2.el6                  cpp.x86_64 0:4.4.7-17.el6
  glibc-devel.x86_64 0:2.12-1.192.el6                glibc-headers.x86_64 0:2.12-1.192.el6
  kernel-headers.x86_64 0:2.6.32-642.4.2.el6         libgomp.x86_64 0:4.4.7-17.el6
  libstdc++-devel.x86_64 0:4.4.7-17.el6              mpfr.x86_64 0:2.4.1-6.el6
  ppl.x86_64 0:0.10.2-11.el6

Dependency Updated:
  glibc.x86_64 0:2.12-1.192.el6    glibc-common.x86_64 0:2.12-1.192.el6  libgcc.x86_64 0:4.4.7-17.el6
  libstdc++.x86_64 0:4.4.7-17.el6  tzdata.noarch 0:2016f-1.el6

Complete!
```

图 3-25　yum 方式安装软件

wget 工具的使用非常简单，通过“wget 下载地址”命令即可将文件下载到当前目录下，并以下载地址中包含的文件名来保存。示例命令如下。

```
[root@localhost ~]#wget http://nginx.org/download/nginx-1.10.1.tar.gz
```

当命令执行后，文件就会被下载到当前目录下，并以 nginx-1.10.1.tar.gz 文件名保存。

脚下留心：时间和时区问题

在 Linux 系统安装完成后，可以通过 date 命令查看当前服务器的时间。在默认情况下，操作系统的时间是读取的计算机主板 BIOS 中的时间，而 VMware 虚拟机能够自动设置 BIOS 时间为物理机时间，因此在安装后，CentOS 的系统时间和物理机 Windows 中显示的时间是相同的。

需要注意的是，在安装系统时，有一个“系统时钟使用 UTC 时间”复选框，在前面讲解的步骤中是取消选中的，这是因为物理机 Windows 系统并没有使用 UTC 时间，如果选中此项，则会因为时区不同导致虚拟机显示的系统时间与物理机不一致。

VMware 虚拟机提供了方便的快照功能，但是在还原到以前的快照时，系统时间并不会自动更新成当前时间，因此推荐读者在还原快照后通过 NTP 服务器联网同步最新时间，具体步骤如下。

(1) 通过 yum 安装 ntpdate 工具。

在最小安装的 CentOS 系统中没有 ntpdate，可以使用 yum 进行安装，具体命令如下。

```
[root@localhost ~]#yum -y install ntpdate
```

(2) 更新系统时间。

目前网络中有许多 NTP 服务器可以使用，读者可以通过百度等搜索引擎查找。下面以 ntp.org 提供的中国节点的服务器为例，具体命令和执行结果如下。

```
[root@localhost ~]#ntpdate cn.pool.ntp.org
28 Sep 15:57:58 ntpdate[1215]: step time server 115.28.122.198 offset 1306.825 sec
[root@localhost ~]#date
Wed Sep 28 15:58:15 CST 2016
```

从上述结果可以看出，已经成功连接到 cn.pool.ntp.org 服务器获取到最新的系统时间。然后通过 date 命令查询当前时间，检查时间是否正确。

多学一招：Linux 系统软件安装方式

Linux 系统中常见的软件安装方式有两种，一种是源码方式安装，另一种是利用 RPM 包和软件包管理器进行安装。

- 所谓源码方式安装，就是将“源码文件”编译成“二进制文件”，并复制到“系统指定目录”的过程。该安装方式的优点是软件运行速度和效率非常高，并可以灵活进行配置。缺点是安装过程相对软件包管理器方式烦琐一些。
- 软件包管理器是一些发行版 Linux 中提供的辅助工具。例如，在 Red Hat、CentOS、Fedora 系统中可以使用 yum(全称为 Yellow dog Updater, Modified)管理软件；在 Debain、Ubuntu 系统中可以使用 APT(Advanced Packaging Tools)管理软件。
- CentOS 系统支持 RPM 包，利用 yum 工具可以从指定的服务器自动下载 RPM 包，并且自动处理依赖关系，实现“一键安装”的效果。该安装方式的优点是软件安装非常方便、快速、智能。缺点是软件版本可能会低于官网提供的版本，并且 yum 服务器并不是囊括了所有软件，对于没有的软件只能手动下载安装。

3.2 Linux 环境下安装 Nginx

在上一节中，完成了 Linux 服务器环境的搭建，并安装了必备的软件。本节将开始讲解如何在 Linux 环境中安装 Nginx，认识 Nginx 的目录结构，以及启动、停止或重启 Nginx 服务。完成 Nginx 的部署后，Web 服务器就搭建完成，在 Windows 系统中可以通过浏览器访问 Web 服务器中的网页。

3.2.1 获取 Nginx

Nginx 在官方网站提供了软件下载。目前 Nginx 发布了 3 种类型的版本，分别为 Mainline version(开发版)、Stable version(稳定版)和 Legacy versions(早期版本)，每种类型的版本中又提供了 Linux 版本和 Windows 版本，如图 3-26 所示。

本书基于稳定版 1.10.1 进行讲解，单击图 3-26 所示的 Stable version 标题下第 2 列的 nginx-1.10.1 可以下载 Nginx 的源码包，下载后上传到 Linux 服务器上即可。

在操作时，读者可以在物理机 Windows 系统上进行下载，然后通过 Xftp 工具将文件上传到 Linux 服务器，也可以从浏览器上复制下载文件的 URL 地址后在服务器中使用 wget 命令进行下载。

下载后的 Nginx 文件是一个.tar.gz 格式的压缩包，在 Linux 系统中使用 tar -zxvf 命令进行解压缩，其中选项 z 表示压缩格式为 gzip，x 表示解压缩，v 表示显示解压过程，f 表示对

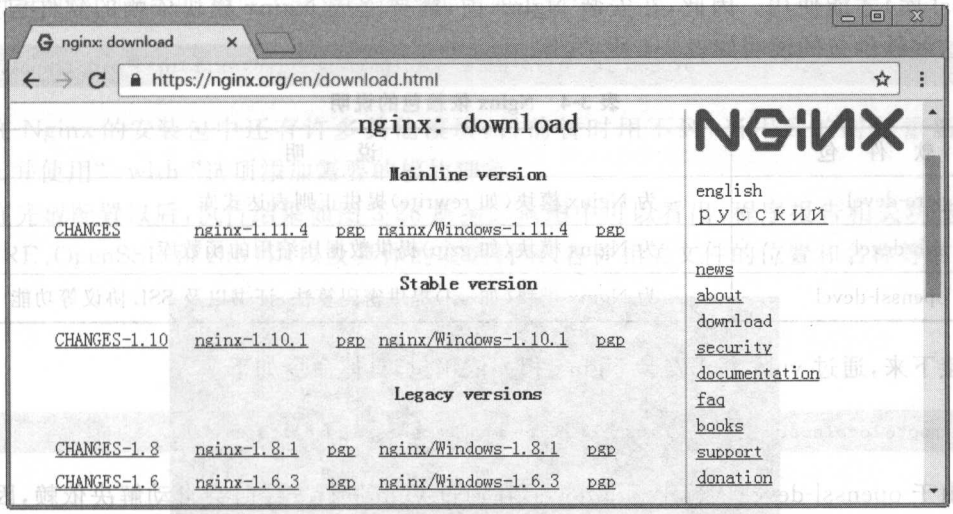


图 3-26　获取 Nginx

文件进行操作。

```
[root@localhost ~]#tar -zxvf nginx-1.10.1.tar.gz
```

上述命令执行后，文件就被解压缩到当前目录下的 nginx-1.10.1 目录中。使用 cd 命令切换到该目录，然后使用 ls 命令查看该目录下的文件，具体如下。

```
[root@localhost ~]#cd nginx-1.10.1
[root@localhost nginx-1.10.1]#ls
auto      CHANGES.ru   configure  html        man      src
CHANGES   conf         contrib    LICENSE     README
```

关于上述目录结构的具体介绍如下。

- src 目录：存放 Nginx 的源代码。
- man 目录：存放 Nginx 的帮助文档。
- html 目录：存放默认网站文件。
- contrib 目录：存放其他机构或组织贡献的文档资料。
- conf 目录：存放 Nginx 服务器的配置文件。
- auto 目录：存放大量的脚本文件，和 configure 脚本程序相关。
- configure 文件：Nginx 自动安装脚本，用于检查环境，生成编译代码需要的 makefile 文件。
- CHANGES、CHANGES. ru、LICENSE 和 README 都是 Nginx 服务器的相关文档资料。

3.2.2　编译安装 Nginx

1. 安装依赖包

由于 Nginx 中的功能是模块化的，而模块又依赖于一些软件包（如 pcre 库、zlib 库和

openssl 库)才能使用。因此,在安装 Nginx 前,需要完成 Nginx 模块依赖的软件包安装。关于这些软件包的说明如表 3-4 所示。

表 3-4 Nginx 依赖包的说明

软 件 包	说 明
pcre-devel	为 Nginx 模块(如 rewrite)提供正则表达式库
zlib-devel	为 Nginx 模块(如 gzip)提供数据压缩用的函数库
openssl-devel	为 Nginx 模块(如 ssl)提供密码算法、证书以及 SSL 协议等功能

接下来,通过 yum 方式安装 Nginx 相关依赖包,具体命令如下。

```
[root@localhost ~]#yum -y install pcre-devel openssl-devel
```

由于 openssl-devel 依赖于 zlib-devel,在通过 yum 进行安装时会自动解决依赖,因此上述命令中省略了 zlib-devel。安装完成后,如图 3-27 所示。

```
Installed:
  openssl-devel.x86_64 0:1.0.1e-48.el6_8.3          pcre-devel.x86_64 0:7.8-7.el6

Dependency Installed:
  keyutils-libs-devel.x86_64 0:1.4-5.el6            krb5-devel.x86_64 0:1.10.3-57.el6
  libcom_err-devel.x86_64 0:1.41.12-22.el6          libselinux-devel.x86_64 0:2.0.94-7.el6
  libsepol-devel.x86_64 0:2.0.41-4.el6              zlib-devel.x86_64 0:1.2.3-29.el6

Dependency Updated:
  openssl.x86_64 0:1.0.1e-48.el6_8.3

Complete!
[root@localhost ~]#
```

图 3-27 安装依赖包

2. Nginx 的编译安装

Nginx 安装的准备工作完成后,接下来使用源码方式完成 Nginx 的安装,具体步骤如下。

(1) 切换到 nginx 解压目录。

```
[root@localhost ~]#cd nginx-1.10.1
```

(2) 配置 Nginx 的编译选项,指定 Nginx 的安装目录。

```
[root@localhost nginx-1.10.1]#./configure \
--prefix=/usr/local/nginx \
--with-http_ssl_module
```

在上述命令中,源码安装的第一步./configure 用于对即将安装的软件进行配置,检查当前的环境是否满足安装软件(Nginx)的依赖关系。其中,--prefix 选项用于设置 Nginx 的安装目录,默认值是/usr/local/nginx,因此也可以省略此选项或指定到其他位置;--with-http_ssl_module 选项用于设置在 Nginx 中允许使用 http_ssl_module 模块的相关功能。

值得一提的是,第 1、2 行末尾的"\"表示当前命令没有结束需要换到下一行书写,直到

第 3 行没有“\”时结束，因此上述命令也等价于如下命令。

```
./configure --prefix=/usr/local/nginx --with-http_ssl_module
```

在 Nginx 的安装包中还有许多其他模块，目前暂时用不到，当用到的时候重新编译 Nginx 并使用“--with-”选项添加需要的模块即可。

在完成配置以后，执行结果如图 3-28 所示。从图中可以看出，程序报告相关功能使用了 PCRE、OpenSSL 和 zlib 库，以及具体的安装目录、各种相关文件的位置和名称等信息。

```
creating objs/Makefile

Configuration summary
  + using system PCRE library
  + using system OpenSSL library
  + md5: using OpenSSL library
  + sha1: using OpenSSL library
  + using system zlib library

  nginx path prefix: "/usr/local/nginx"
  nginx binary file: "/usr/local/nginx/sbin/nginx"
  nginx modules path: "/usr/local/nginx/modules"
  nginx configuration prefix: "/usr/local/nginx/conf"
  nginx configuration file: "/usr/local/nginx/conf/nginx.conf"
  nginx pid file: "/usr/local/nginx/logs/nginx.pid"
  nginx error log file: "/usr/local/nginx/logs/error.log"
  nginx http access log file: "/usr/local/nginx/logs/access.log"
  nginx http client request body temporary files: "client_body_temp"
  nginx http proxy temporary files: "proxy_temp"
  nginx http fastcgi temporary files: "fastcgi_temp"
  nginx http uwsgi temporary files: "uwsgi_temp"
  nginx http scgi temporary files: "scgi_temp"

[root@localhost nginx-1.10.1]#
```

图 3-28　Nginx 编译前的配置

(3) 通过 make 命令编译和安装 Nginx。

```
[root@localhost nginx-1.10.1]#make && make install
```

在上述命令中，&& 用于连接两个命令，根据前面(左边)命令的返回值决定是否执行后面(右边)的命令，只有前面的命令成功时才会执行后面的命令。这种方式可以简化手动操作的过程，实际上读者也可以分别单独执行 make 和 make install 两个命令。

完成 Nginx 的编译和安装之后的效果如图 3-29 所示。

```
cp conf/uwsgi_params \
                '/usr/local/nginx/conf/uwsgi_params.default'
test -f '/usr/local/nginx/conf/scgi_params' \
                || cp conf/scgi_params '/usr/local/nginx/conf'
cp conf/scgi_params \
                '/usr/local/nginx/conf/scgi_params.default'
test -f '/usr/local/nginx/conf/nginx.conf' \
                || cp conf/nginx.conf '/usr/local/nginx/conf/nginx.conf'
cp conf/nginx.conf '/usr/local/nginx/conf/nginx.conf.default'
test -d '/usr/local/nginx/logs' \
                || mkdir -p '/usr/local/nginx/logs'
test -d '/usr/local/nginx/logs' \
                || mkdir -p '/usr/local/nginx/logs'
test -d '/usr/local/nginx/html' \
                || cp -R html '/usr/local/nginx'
test -d '/usr/local/nginx/logs' \
                || mkdir -p '/usr/local/nginx/logs'
make[1]: Leaving directory `/root/nginx-1.10.1'
[root@localhost nginx-1.10.1]#
```

图 3-29　编译和安装 Nginx

多学一招：devel 包和非 devel 的区别

Linux 中的某些软件包具有 devel 包和非 devel 包两种形式，如 zlib 和 zlib-devel。那么两者有什么区别呢？非 devel 包就是普通的软件包，而 devel 包则一般会包括头文件、静态库甚至源码。若仅仅使用这些软件，则仅安装非 devel 包即可，但若在开发时需要用到这些软件包中的共享库，就需要安装 devel 包。

通常在使用 yum -y install 安装 devel 包时，服务器会自动先安装非 devel 包，然后再安装 devel 包。因此，当同时需要使用两种包时，可以在 yum 命令中省略非 devel 包的书写。

3.2.3 Nginx 的启动与停止

1. 启动 Nginx

Nginx 安装完成后，切换到 Nginx 安装目录中的 sbin 目录，通过执行该目录下 Nginx 编译后的二进制文件即可启动程序。具体命令如下。

```
[root@localhost ~]#cd /usr/local/nginx/sbin
[root@localhost sbin]#./nginx
```

执行上述操作后，若成功启动 Nginx，则程序没有任何提示。使用 ps 命令可以查看 Nginx 的运行状态，具体命令如下。

```
[root@localhost ~]#ps aux | grep nginx
```

以上执行了 ps aux 和 grep nginx 两个命令。通过这组命令，可以从所有的进程中找到是否存在 Nginx 进程。执行后的效果如图 3-30 所示。

```
[root@localhost ~]# ps aux | grep nginx
root       8170  0.0  0.2  44724  1168 ?        Ss   01:58   0:00 nginx: master process ./nginx
nobody     8171  0.0  0.3  45148  1764 ?        S    01:58   0:00 nginx: worker process
root       8173  0.0  0.1 103328   888 pts/2    S+   01:58   0:00 grep nginx
[root@localhost ~]#
```

图 3-30 查看 Nginx 进程

在图 3-30 的输出结果中，前 2 行分别是 Nginx 主进程(master process)和工作进程(worker process)，第 3 行是 grep nginx 命令。当看到这两个 Nginx 进程时，说明 Nginx 已经启动。从第 1 列可以看出，Nginx 主进程以 root 用户运行，而工作进程以 nobody 用户运行；第 2 列显示了两个进程的 ID(即 PID)，分别为 8170 和 8171。

2. 停止 Nginx 服务

当需要停止 Nginx 服务时，有多种停止方式，可以根据需求采取不同的方式，具体如下。

1) 立即停止服务

Nginx 程序允许传递选项-s 表示发送信号到主进程，如果后面跟上 stop 表示停止服务。

```
[root@localhost sbin]# ./nginx -s stop
```

2）从容停止服务

上面讲解的 stop 方式是立即停止 Nginx 服务，无论当前工作进程是否正在处理工作。而 Nginx 提供的从容停止方式 quit，是在完成当前工作任务后再停止。

```
[root@localhost sbin]# ./nginx -s quit
```

3）通过 kill 或 killall 命令杀死进程

Linux 中提供了 kill 和 killall 命令可以杀死进程，从而让指定进程停止运行。

方式一：

```
[root@localhost ~]#kill Nginx 主进程的 PID
```

方式二：

```
[root@localhost ~]#killall nginx
```

在 Nginx 中，除了以上提到的启动和停止 Nginx 服务的操作，还有一些其他常用的命令，如表 3-5 所示。

表 3-5　Nginx 常用命令

命　令	说　明
nginx -s reload	在 Nginx 已经启动的情况下重新加载配置文件(平滑重启)
nginx -s reopen	重新打开日志文件
nginx -c /特定目录/nginx. conf	以特定目录下的配置文件启动 Nginx
nginx -t	检测当前配置文件是否正确
nginx -t -c /特定目录/nginx. conf	检测特定目录下的 Nginx 配置文件是否正确
nginx -v	显示版本信息
nginx -V	显示版本信息和编译选项

3. 查看端口号占用

在默认情况下，Nginx 启动后会监听 80 端口，从而提供 HTTP 访问，如果 80 端口已经被占用则会启动失败。例如，在 Nginx 已经启动的情况下再次执行 Nginx 启动的命令就会报错。在 Linux 系统中，可以使用 netstat -tlnp 命令查看端口号占用的情况，如图 3-31 所示。

```
[root@localhost init.d]# netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address     State       PID/Program name
tcp        0      0 0.0.0.0:80         0.0.0.0:*           LISTEN      8170/nginx
tcp        0      0 0.0.0.0:22         0.0.0.0:*           LISTEN      1254/sshd
tcp        0      0 127.0.0.1:25       0.0.0.0:*           LISTEN      1335/master
tcp        0      0 :::22              :::*                LISTEN      1254/sshd
tcp        0      0 ::1:25             :::*                LISTEN      1335/master
```

图 3-31　查看端口号

从图 3-31 中可以看出，Nginx 的主进程正在监听 TCP 协议 80 端口，说明 Nginx 目前已经启动。另外，netstat 命令的 4 个选项 t、l、n、p 分别表示查看 tcp 协议、查看监听服务、不解析名称以及显示进程名和 PID。

3.2.4 访问测试

在将基于 Nginx 的 Web 服务器部署完成后，就可以在客户端通过浏览器进行访问。在默认情况下 CentOS 系统开启了 iptables 防火墙，Nginx 为提供 HTTP 访问所监听的 80 端口是被阻止访问的。为了能让其他计算机通过浏览器访问 Web 服务器，就需要开放 80 端口。

下面通过配置 iptables 防火墙，实现开放 80 端口的访问，具体命令如下。

```
[root@localhost ~]#iptables -I INPUT -p tcp --dport 80 -j ACCEPT
```

上述命令表示防火墙对于来自外部访问的请求，如果是 TCP 协议和 80 端口，则接受访问。关于以上命令各个参数的含义和详细说明如表 3-6 所示。

表 3-6 iptables 防火墙参数说明

参 数	说 明
-I INPUT	表示在 INPUT(外部访问规则)中插入一条规则
-p tcp	指定数据包匹配的协议(tcp、udp、icmp 等)，这里指定 tcp 协议
--dport 80	用于指定数据包匹配的目标端口号，这里指定 80 端口
-j ACCEPT	指定对数据包的处理操作(ACCEPT、DROP、REJECT、REDIRECT 等)，这里指定 ACCEPT 操作

执行完上述命令后，若没有任何提示，则表示执行成功。为了进一步确认修改是否成功，可以通过查看防火墙的状态进行确认。具体命令如下。

```
[root@localhost ~]#service iptables status
```

执行上述操作后，输出结果如图 3-32 所示。在输出结果中，小标题 Chain INPUT(policy ACCEPT)下面 num 为 1 的一行中，最右边的 tcp dpt:80 说明这一行就是前面添加的规则。

```
[root@localhost ~]# iptables -I INPUT -p tcp --dport 80 -j ACCEPT
[root@localhost ~]# service iptables status
Table: filter
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:80
2    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
3    ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0
4    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
5    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           state NEW tcp dpt:22
6    REJECT     all  --  0.0.0.0/0            0.0.0.0/0           reject-with icmp-host-prohibited

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination
1    REJECT     all  --  0.0.0.0/0            0.0.0.0/0           reject-with icmp-host-prohibited

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

[root@localhost ~]#
```

图 3-32 查看防火墙状态

需要注意的是，上述操作只是临时生效并没有保存，在系统重启或 iptables 服务重启后会恢复原来的规则。如果需要保存到防火墙规则中，则执行以下命令即可。

```
[root@localhost ~]#service iptables save
```

接下来可以重启 iptables 服务进行测试，如果前面添加的规则仍然存在则说明保存成功。

```
[root@localhost ~]#service iptables restart
```

在防火墙规则生效之后，即可在物理机（Windows 系统）中通过浏览器进行访问，访问的 URL 地址为“http://服务器的 IP 地址”。如果看到如图 3-33 所示的效果，则表示访问成功。

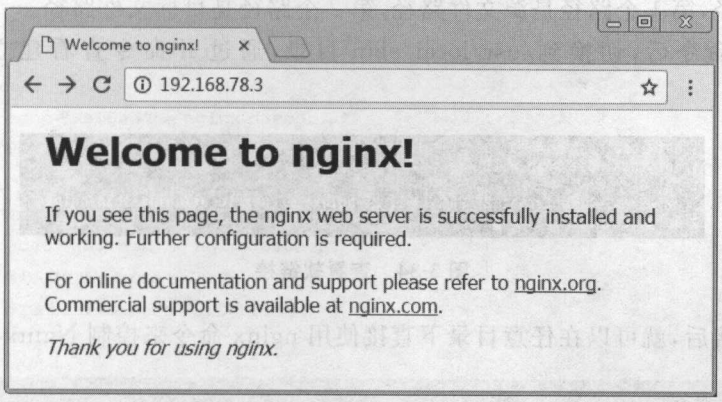


图 3-33　访问测试

3.2.5　后续操作

在 Nginx 安装成功后，为了更方便地使用和操作 Nginx 服务，下面对 Nginx 如何添加到环境变量、系统服务和开机自启动的实现进行讲解。

1. 添加到环境变量

在每次启动或停止 Nginx 服务时，都必须输入 Nginx 的安装目录，麻烦又烦琐。为了更方便地使用 Nginx，可以将 Nginx 添加到环境变量中，这样不论当前处于什么目录下，都可以直接使用命令的方式操作 Nginx。

在 Linux 系统中，当执行一个命令时（如执行 nginx），会自动搜索环境变量中指定的目录中是否存在相应的程序。使用 echo 命令输出 $PATH 的值可以查看当前环境变量，具体输出结果如下。

```
[root@localhost ~]#echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

在上述结果中，环境变量是由冒号“:”分隔多个目录组成的字符串。当系统自动搜索环境变量时，会优先搜索最左边的路径（即/usr/local/sbin），然后依次向右搜索，找到后就会

执行并停止搜索。如果在所有目录中都没有找到，则会提示 command not found(命令未找到)。

这些环境变量目录每个都有特定的用途，其中/bin 和/sbin 放置常用程序，sbin 表示需要管理员权限；/usr/bin 和/usr/sbin 放置一些工具软件的可执行程序；/usr/local/bin 和/usr/local/sbin 放置用户自行安装的可执行程序。因此，推荐将 Nginx 放入/usr/local/sbin 目录中。

接下来利用软链接将 nginx 程序链接到/usr/local/sbin 目录中，从而创建 nginx 命令。具体操作如下。

```
[root@localhost ~]# ln -s /usr/local/nginx/sbin/nginx /usr/local/sbin/nginx
```

上述命令中，ln 用于创建链接，选项-s 表示创建软链接，类似 Windows 中的快捷方式，后面跟两个路径，第 1 个路径是源文件路径，第 2 个路径是目标文件路径。

执行上述命令后，切换到/usr/local/sbin 目录，通过 ll 命令查看建立的软链接，如图 3-34 所示。

```
[root@localhost ~]# cd /usr/local/sbin
[root@localhost sbin]# ll
total 0
lrwxrwxrwx. 1 root root 27 Sep 30 11:21 nginx -> /usr/local/nginx/sbin/nginx
[root@localhost sbin]#
```

图 3-34 查看软链接

创建软链接后，就可以在任意目录下直接使用 nginx 命令来控制 Nginx 服务。

```
#停止 Nginx 服务
[root@localhost ~]#nginx -s quit
#启动 Nginx 服务
[root@localhost ~]#nginx
#重新载入配置
[root@localhost ~]#nginx -s reload
```

2. 添加到系统服务

在前面的学习中，许多 Linux 系统服务都可以通过 service 命令进行控制。例如，控制网络重新加载使用 service network reload 命令，控制防火墙关闭使用 service iptables stop 命令。其中 service 命令的第 1 个参数是服务的名称，第 2 个参数是具体的操作(如 start、stop、restart、reload 等)。

在 CentOS 系统中，service 命令实际上是调用了/etc/init.d 目录下的 shell 脚本，也就是说，如下两行命令其实是等价的。

```
#直接执行脚本
[root@localhost ~]#/etc/init.d/network restart
#通过 service 命令执行脚本
[root@localhost ~]#service network restart
```

在上述命令中，network 是 shell 脚本的文件名，restart 是传递给脚本的参数。因此，将

Nginx 添加到系统服务中时，只需要在/etc/init.d 中编写一个文件名为 nginx 的 shell 脚本即可。

接下来执行 vi /etc/init.d/nginx 命令编写一个 shell 脚本实现 Nginx 服务管理，提供 start、stop、reload、restart 4 个参数，具体代码如下。

```
1   #!/bin/bash
2   DAEMON=/usr/local/nginx/sbin/nginx
3   case "$1" in
4     start)
5         echo "Starting nginx daemon..."
6         $DAEMON && echo "SUCCESS"
7     ;;
8     stop)
9         echo "Stopping nginx daemon..."
10        $DAEMON -s quit && echo "SUCCESS"
11    ;;
12    reload)
13        echo "Reloading nginx daemon..."
14        $DAEMON -s reload && echo "SUCCESS"
15    ;;
16    restart)
17        echo "Restarting nginx daemon..."
18        $DAEMON -s quit
19        $DAEMON && echo "SUCCESS"
20    ;;
21    *)
22        echo "Usage: service nginx {start|stop|restart|reload}"
23        exit 2
24    ;;
25  esac
```

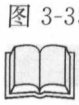
上述第 1 行代码表示下面编写的脚本是在 bash shell 环境下运行并执行的，第 2 行代码用于定义一个变量 DAEMON，用于保存 Nginx 程序的路径，在使用变量时需要在名称前加上$符号。

第 3～25 行代码是 case 语句，“$1”是 shell 预定义的参数变量，用于接收传入的参数，数字 1 表示用户输入的第 1 参数。case 语句将根据传入的参数执行相应的操作，当没有合适的匹配项时，则执行第 21～24 行的默认操作。第 23 行的 exit 用于退出脚本并返回状态代码，数值大于 0 表示执行失败。

完成上述脚本编写并保存后，使用如下命令为 nginx 脚本添加可执行权限。

```
[root@localhost ~]#chmod +x /etc/init.d/nginx
```

下面使用 service 命令测试 nginx 服务是否可以进行管理，具体命令和运行结果如图 3-35 所示。



多学一招：shell 脚本相关语法知识

shell 脚本是一种预先写好的脚本，可以将一系列命令放入一个文件中，方便一次性执

```
[root@localhost init.d]# service nginx stop
Stopping nginx daemon...
SUCCESS
[root@localhost init.d]# service nginx start
Starting nginx daemon...
SUCCESS
[root@localhost init.d]# service nginx restart
Restarting nginx daemon...
SUCCESS
[root@localhost init.d]# service nginx reload
Reloading nginx daemon...
SUCCESS
[root@localhost init.d]#
```

图 3-35 添加 nginx 系统服务

行。要想使用 shell 进行编程，首先要了解它的基本语法结构，下面将介绍几种常用语法。

(1) shell 语法格式。

在 shell 脚本文件的开头，需要使用特殊表示符号#!定义解释此脚本的 shell 路径。例如，该脚本使用 bash 环境执行，则在 shell 脚本开头编写如下代码。

```
#! /bin/bash
```

(2) case 语句。

case 语句用于多重分支语句匹配的情况，具体语法格式如下。

```
case $变量名 in
    模式 1)
        命令序列 1
    ;;
    模式 2)
        命令序列 2
    ;;
    * )
        默认执行的命令序列
    ;;
esac
```

上述语法中，case 语句首行必须以 case 开始，in 结尾，中间的变量表示用户输入的字符；每个模式必须以右括号“)”结束，双分号“;;”结束命令序列，且匹配模式中可以使用方括号表示一个连续的范围，如[0-9]，使用竖杠符号“|”表示“或”；最后的“*)”是默认模式，当使用前面的各种模式均无法匹配该变量时，将执行“*)”后的命令序列；最后 case 语句必须以 esac 结束。

3. 设置开机自启动

对于一个要经常使用的服务器而言，每次开机后，都需要用户手动开启一些服务较为麻烦。接下来，将通过 chkconfig 命令完成 Nginx 开机自启动的功能。chkconfig 命令的语法格式如下所示：

```
chkconfig [--add][--del][--list][系统服务]
```

在上述语法中，--add 用于增加指定的系统服务(如 nginx)，设置该服务为开机自启

动；--del 用于删除指定的系统服务，取消该服务的开机自启动；--list 用于列出系统所有的服务启动情况，如图 3-36 所示。

```
[root@localhost ~]# chkconfig --list
auditd          0:off   1:off   2:on    3:on    4:on    5:on    6:off
blk-availability        0:off   1:on    2:on    3:on    4:on    5:on    6:off
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
ip6tables       0:off   1:off   2:on    3:on    4:on    5:on    6:off
iptables        0:off   1:off   2:on    3:on    4:on    5:on    6:off
kdump           0:off   1:off   2:off   3:on    4:on    5:on    6:off
lvm2-monitor    0:off   1:on    2:on    3:on    4:on    5:on    6:off
mdmonitor       0:off   1:off   2:on    3:on    4:on    5:on    6:off
netconsole      0:off   1:off   2:off   3:off   4:off   5:off   6:off
netfs           0:off   1:off   2:off   3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
nfs-rdma        0:off   1:off   2:off   3:off   4:off   5:off   6:off
postfix         0:off   1:off   2:on    3:on    4:on    5:on    6:off
rdisc           0:off   1:off   2:off   3:off   4:off   5:off   6:off
rdma            0:off   1:off   2:off   3:off   4:off   5:off   6:off
restorecond     0:off   1:off   2:off   3:off   4:off   5:off   6:off
rsyslog         0:off   1:off   2:on    3:on    4:on    5:on    6:off
saslauthd       0:off   1:off   2:off   3:off   4:off   5:off   6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
udev-post       0:off   1:on    2:on    3:on    4:on    5:on    6:off
[root@localhost ~]#
```

图 3-36　查看服务启动情况

若要将 Nginx 服务实现开机启动，就需要在/etc/init.d/nginx 脚本文件中添加对 chkconfig 的支持。使用编辑器打开 nginx 脚本文件，在第 2 行的位置新增如下内容。

```
#chkconfig: 35 85 15
```

上述代码中，冒号后的 35 表示 Nginx 服务允许的启动级别是 3 和 5，若暂不开启任何启动级别，可以使用一个横线“-”代替。85 和 15 用于设置 Nginx 服务的启动(S)顺序和关闭(K)顺序，即 S85 和 K15。数值小的先执行，数值大的后执行。该值是用户自定义的，取值范围在 0～99 之间。对于 Nginx 服务，推荐 S 值较大，K 值较小(即晚启动，早关闭)。

接下来使用 chkconfig 命令执行添加 Nginx 服务自启动操作，具体命令如下。

```
[root@localhost ~]#chkconfig --add nginx
```

执行命令后利用 chkconfig --list 命令查看添加后的效果，如图 3-37 所示。从图中可以看出，nginx 服务已经添加成功，且 7 个启动级别中只有 3 和 5 是 on(开启)状态，其他级别是 off(关闭)状态。读者可以重新启动 Linux 系统，测试 Nginx 服务是否能够开机自启动。

```
[root@localhost ~]# chkconfig --list
auditd          0:off   1:off   2:on    3:on    4:on    5:on    6:off
blk-availability        0:off   1:on    2:on    3:on    4:on    5:on    6:off
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
ip6tables       0:off   1:off   2:on    3:on    4:on    5:on    6:off
iptables        0:off   1:off   2:on    3:on    4:on    5:on    6:off
kdump           0:off   1:off   2:off   3:on    4:on    5:on    6:off
lvm2-monitor    0:off   1:on    2:on    3:on    4:on    5:on    6:off
mdmonitor       0:off   1:off   2:on    3:on    4:on    5:on    6:off
netconsole      0:off   1:off   2:off   3:off   4:off   5:off   6:off
netfs           0:off   1:off   2:off   3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
nfs-rdma        0:off   1:off   2:off   3:off   4:off   5:off   6:off
nginx           0:off   1:off   2:off   3:on    4:off   5:on    6:off
postfix         0:off   1:off   2:on    3:on    4:on    5:on    6:off
rdisc           0:off   1:off   2:off   3:off   4:off   5:off   6:off
rdma            0:off   1:off   2:off   3:off   4:off   5:off   6:off
restorecond     0:off   1:off   2:off   3:off   4:off   5:off   6:off
rsyslog         0:off   1:off   2:on    3:on    4:on    5:on    6:off
saslauthd       0:off   1:off   2:off   3:off   4:off   5:off   6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
udev-post       0:off   1:on    2:on    3:on    4:on    5:on    6:off
[root@localhost ~]#
```

图 3-37　查看 nginx 服务启动情况

多学一招：查看服务启动和关闭顺序

在 CentOS 系统中，服务的启动和关闭顺序是在“/etc/rc 数字.d”目录中保存的，“数字”表示运行级别。以启动级别 3 为例，进入/etc/rc3.d 目录下查看文件列表，如图 3-38 所示。

```
[root@localhost ~]# cd /etc/rc3.d
[root@localhost rc3.d]# ll
total 0
lrwxrwxrwx. 1 root root 19 Sep 30 09:45 K10saslauthd -> ../init.d/saslauthd
lrwxrwxrwx. 1 root root 18 Sep 30 09:46 K61nfs-rdma -> ../init.d/nfs-rdma
lrwxrwxrwx. 1 root root 21 Sep 30 09:45 K87restorecond -> ../init.d/restorecond
lrwxrwxrwx. 1 root root 20 Sep 30 09:45 K89netconsole -> ../init.d/netconsole
lrwxrwxrwx. 1 root root 15 Sep 30 09:45 K89rdisc -> ../init.d/rdisc
lrwxrwxrwx. 1 root root 14 Sep 30 09:46 K95rdma -> ../init.d/rdma
lrwxrwxrwx. 1 root root 22 Sep 30 09:46 S02lvm2-monitor -> ../init.d/lvm2-monitor
lrwxrwxrwx. 1 root root 19 Sep 30 09:46 S08ip6tables -> ../init.d/ip6tables
lrwxrwxrwx. 1 root root 18 Sep 30 09:45 S08iptables -> ../init.d/iptables
lrwxrwxrwx. 1 root root 17 Sep 30 09:45 S10network -> ../init.d/network
lrwxrwxrwx. 1 root root 16 Sep 30 09:46 S11auditd -> ../init.d/auditd
lrwxrwxrwx. 1 root root 17 Sep 30 09:45 S12rsyslog -> ../init.d/rsyslog
lrwxrwxrwx. 1 root root 19 Sep 30 09:45 S15mdmonitor -> ../init.d/mdmonitor
lrwxrwxrwx. 1 root root 26 Sep 30 09:46 S25blk-availability -> ../init.d/blk-availability
lrwxrwxrwx. 1 root root 15 Sep 30 09:45 S25netfs -> ../init.d/netfs
lrwxrwxrwx. 1 root root 19 Sep 30 09:45 S26udev-post -> ../init.d/udev-post
lrwxrwxrwx. 1 root root 15 Sep 30 09:46 S50kdump -> ../init.d/kdump
lrwxrwxrwx. 1 root root 14 Sep 30 09:46 S55sshd -> ../init.d/sshd
lrwxrwxrwx. 1 root root 17 Sep 30 09:45 S80postfix -> ../init.d/postfix
lrwxrwxrwx. 1 root root 15 Sep 30 15:55 S85nginx -> ../init.d/nginx
lrwxrwxrwx. 1 root root 15 Sep 30 09:46 S90crond -> ../init.d/crond
lrwxrwxrwx. 1 root root 11 Sep 30 09:45 S99local -> ../rc.local
[root@localhost rc3.d]#
```

图 3-38 查看服务启动和关闭顺序

在图 3-38 文件列表中，所有的文件都是/etc/init.d 中的服务脚本的软链接，其文件名格式是由 S 或 K、顺序值、服务名称 3 部分组成的。对于底层的系统服务，通常 S 值小 K 值大（即早启动、晚关闭），而对于普通服务，通常 S 值大 K 值小（即晚启动、早关闭）。在图 3-38 中可以看到 nginx 服务顺序为 S85。而 K15 位于除了 3 和 5 的其他级别中，表示在关机或重启时自动关闭 Nginx。

在系统开机时，会按照 S 的顺序从小到大调用脚本，并传递参数 start；在系统关闭时，会按照 K 的顺序从小到大调用脚本，并传递参数 stop。若需要修改顺序时，可以修改服务脚本中“#chkconfig:”指定的顺序，然后使用 chkconfig 命令重新添加服务。

3.3 Windows 环境下使用 Nginx

Nginx 目前提供 Linux 和 Windows 两种环境的版本，目前 Windows 版本在生产环境并不常用，适合简单的练习使用。下面讲解如何在 Windows 环境下使用 Nginx。

1. 下载并解压

从 Nginx 官网（http://nginx.org）下载 Stable version（稳定版）的 Windows 版本 Nginx，文件名为 nginx-1.10.1.zip，下载后解压到目录 C:\web\nginx1.10 中（也可以是其他路径），如图 3-39 所示。

图 3-39 是 Windows 版本 Nginx 的目录结构，其中 conf 目录存放 Nginx 的配置文件，contrib 目录存放一些实用工具，docs 目录存放文档，html 目录存放网站默认目录，logs 目

图 3-39　解压 Nginx

录存放日志，temp 目录存放临时文件。

2. 启动与停止 Nginx 服务

Windows 版本的 Nginx 的启动非常简单，双击 nginx.exe 启动程序即可。在 Nginx 启动后，可以利用 cmd 命令行工具执行 tasklist 命令查看进程，如图 3-40 所示。

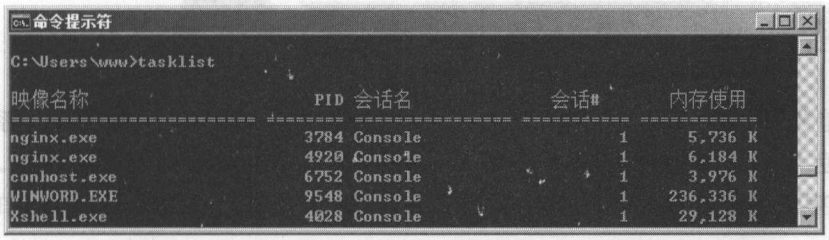


图 3-40　查看进程

需要注意的是，Nginx 默认监听的端口号是 80，因此应确保 80 端口没有被其他程序占用，如 Micorsoft IIS、Apache HTTP Server 或 Tomcat 等 Web 服务器软件。在命令行中执行 netstat -ano 命令可以查看 80 端口当前是否已经占用，如图 3-41 所示。

图 3-41　查看端口占用

若要停止 Nginx 服务，则需要以管理员身份打开 cmd 命令窗口，切换到 nginx.exe 所在的目录，执行以下命令即可。nginx.exe 程序的参数与 Linux 版本相同。

```
C:\web\nginx1.10>nginx.exe -s stop
```

3. 访问测试

Nginx 成功启动后，就可以在浏览器输入 http://localhost 或 http://127.0.0.1 进行访问。如果看到如图 3-42 所示的内容，说明在 Windows 环境下 Nginx 已经启动。

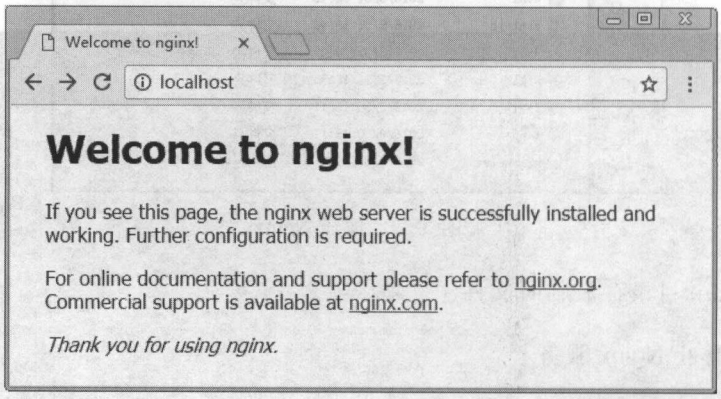


图 3-42 访问测试

本章小结

本章首先介绍 Linux 服务器环境的搭建，然后重点讲解如何在 Linux 系统中安装 Nginx，搭建 Web 服务器环境，最后讲解 Windows 环境下 Nginx 的安装与使用。通过本章的学习，希望读者能够熟练掌握 Nginx 的安装步骤，以及 Linux 服务器环境下的一些基本配置。

课后练习

一、填空题

1. 在 Linux 系统中，创建软链接的命令和选项是_____。
2. 执行 nginx 命令的_____参数可以在更改配置文件后平滑重启。

二、判断题

1. Nginx 平滑重启的命令是 nginx -t。 ( )
2. 通过 service 命令执行的服务脚本存放在 var 目录中。 ( )

三、选择题

1. netstat -tlnp 命令用于查看端口号占用的情况，其中选项 l 表示( )。
   A. tcp 协议　　B. 查看监听服务
   C. 不解析名称　　D. 显示进程名和 PID
2. 下列选项中，用来查看当前防火墙状态的命令是( )。

A. service iptables start　　B. service iptables status

C. service iptables restart　　D. service iptables save

3. 在 VMware 中，下列哪个选项可以查看虚拟机网卡的 MAC 地址(　　)。

A. 网络适配器　　B. CD/DVD(IDE)　　C. 硬盘　　D. 处理器

四、简答题

1. 请简述 VMware 虚拟机提供了几种虚拟网络配置方式及各自的特点。

2. 请简述 devel 包和非 devel 包的区别。

五、操作题

在配置服务器的工作中，crontab 是一个经常使用的命令，用于设置周期性被执行的任务。例如，每小时执行一次脚本、每天清理一次临时文件和缓存、每周备份一次数据等。请尝试在 CentOS 系统中实现每隔 1 分钟更新一次系统时间，从而在 VMware 还原快照后及时地自动更新时间。

第4章 Nginx基本配置

学习目标

- 了解 Nginx 配置文件结构；
- 掌握目录配置和访问控制；
- 熟悉日志文件的配置与切割；
- 掌握虚拟主机的配置。

学习 Nginx 首先一定要了解其配置文件的基本结构、常用指令的含义以及其使用方法，原因在于 Nginx 所有功能的实现，都是通过配置文件的设置来完成的，例如，虚拟主机、反向代理、负载均衡等的实现。本章将针对 Nginx 的基本配置进行详细讲解。

4.1 认识配置文件

Nginx 服务器安装完成后，默认安装时自带的配置文件全部存储在 conf 目录下，并且为了备份还原，每个配置文件都提供了一个以. default 结尾的备份文件。其中，nginx. conf 是 Nginx 默认的主配置文件，所有功能的实现都与此文件的配置相关。下面对该文件的结构以及默认设置指令的含义进行详细介绍。

4.1.1 配置文件结构

打开 nginx. conf 配置文件，从整体结构可以看出，该配置文件主要由以下几部分组成。

```
main
events {...}
http {
    server {
        location {...}
    }
}
```

从上面的结构可以看出，Nginx 的默认主配置文件主要由 main、events、http、server 和 location 5 个块组成，关于各个块的作用，详见表 4-1 所示。并且对于嵌套块（如 http、server、location）中的指令，执行的顺序为从外到内依次执行，内层块中的大部分指令会自动获取外层块指令的值作为默认值，只有某些特殊指令除外。

表 4-1　配置文件结构

块	说　明
main	主要控制 Nginx 子进程所属的用户和用户组、派生子进程数、错误日志位置与级别、pid 位置、子进程优先级、进程对应 CPU、进程能够打开的文件描述符数目等
events	控制 Nginx 处理连接的方式
http	Nginx 处理 http 请求的主要配置块，大多数配置都在这里面进行
server	Nginx 中主机的配置块，可用于配置多个虚拟主机
location	server 中对应目录级别的控制块，可以有多个

在介绍 Nginx 配置文件的基本结构以及各个组成部分的含义后，接下来详细了解一下默认配置指令的具体含义。需要注意的是，在默认的配置文件中，有很多以 # 开始的注释行，Nginx 并不会对其进行解析，该注释行的作用仅用于解释和说明。下面为了便于阅读，去掉配置文件中所有注释后，默认的配置整体展示如下所示。

```
1   worker_processes  1;
2   events {
3       worker_connections  1024;
4   }
5   http {
6       include       mime.types;
7       default_type application/octet-stream;
8       sendfile      on;
9       keepalive_timeout  65;
10      server {
11          listen       80;
12          server_name  localhost;
13          location / {
14              root   html;
15              index  index.html index.htm;
16          }
17          error_page   500 502 503 504  /50x.html;
18          location=/50x.html {
19              root   html;
20          }
21      }
22  }
```

从上述配置可以看出，Nginx 的指令由指令名称和参数组成。例如，第 1 行 worker_processes 指令的参数为 1，第 3 行 worker_connections 指令的参数为 1024 等。当一个指令中含有多个子指令作为参数时，需要使用大括号{ }进行包裹，如 2～4 行配置，且每条指令都以分号“;”结尾。

关于上述 Nginx 默认配置文件中指令的含义，详见表 4-2 所示。其中，在该配置中引入的文件路径，可以是相对路径，也可以是绝对路径。

相对路径的设置如上述第 6 行的配置，表示引入的 mime.types 文件是相对于当前配置文件 nginx.conf 所在的目录/usr/local/nginx/conf。绝对路径的设置就是以 Linux 的根目

录“/”开始的文件路径。例如，可以将 mime.types 文件的引入路径写成如下形式。

```
include    /usr/local/nginx/conf/mime.types;
```

在上述配置中，mime.types 文件的作用是存储文件扩展名与文件类型的映射表。

表 4-2 默认配置指令

指　令	说　明
worker_processes	配置 Nginx 的工作进程数，一般设为 CPU 总核数或者总核数的两倍
worker_connections	配置 Nginx 允许单个进程并发连接的最大请求数
include	用于引入配置文件
default_type	设置默认文件类型
sendfile	默认值为 on，表示开启高效文件传输模式
keepalive_timeout	设置长连接超时时间(单位：秒)
listen	监听端口，默认监听 80 端口
server_name	设置主机域名
root	设置主机站点根目录地址
index	指定默认索引文件
error_page	自定义错误页面

以上就是 nginx.conf 配置文件中默认指令的相关说明，读者了解即可。接下来会在后续的章节中分别对这些指令的使用以及注意事项详细分析讲解。

4.1.2 设置用户和组

在讲解如何配置 Nginx 用户和组之前，先了解一下 Nginx 中用户和组的作用。

Nginx 服务是由一个主进程(master process)和多个工作进程(worker process)组成的。其中，主进程以 root 权限运行，而工作进程在默认情况下以 nobody 用户运行。原因在于 nobody 用户是一个不能登录的账号，有一个专用的 ID，可将每个运行的工作进程隔离出来，这样即使黑客破坏了服务器程序，因其不是 root 用户，也不会影响其他数据。

因此，为工作进程设置的执行用户权限越低，则服务器安全系数越高。

接下来，通过 ps 命令查看当前启动的 Nginx 服务器中主进程(master process)和工作进程(worker process)的用户权限，具体命令如下。

```
[root@localhost ~]#ps aux | grep nginx
```

执行上述命令后，运行结果如图 4-1 所示。从图中可以看出 Nginx 默认有一个主进程(master process)和一个工作进程(worker process)，以及用户和组的分配情况。

Nginx 提供配置用户和组的功能，针对的就是工作进程(worker process)，主要用于对某些操作提供权限。例如，配置日志文件时，主进程创建日志文件后，会以工作进程的用户

```
[root@localhost ~]# ps aux | grep nginx
root      1338  0.0  0.1  44860  1908 ?        Ss   09:16   0:00 nginx: master process /usr/local/nginx/sbin/nginx
nobody    1446  0.0  0.1  45088  1832 ?        S    09:22   0:00 nginx: worker process
root      1451  0.0  0.0 103312   876 pts/0    S+   09:22   0:00 grep nginx
[root@localhost ~]#
```

图 4-1　查看 Nginx 默认用户和组

作为文件所有者，从而使工作进程能够将日志写入指定文件中。

Nginx 提供两种设置用户和组的方式，一种是在安装时通过编译选项进行设置，另一种是修改配置文件。需要注意的是，不论哪种方式在配置之前，都需要提前创建好用户和组。

1. 编译安装配置方式

在./configure 编译安装 Nginx 时的选项中，添加如下两个选项。

```
--user=<user>
--group=<group>
```

在上述选项中，user 用于指定用户名称，group 用于指定用户所在组的名称。

2. 修改配置文件方式

打开 Nginx 的配置文件，找到配置用户和组的指令 user，具体如下。

```
#user  nobody;
```

接下来以用户 nuser 和组 ngroup 为例，修改后的配置如下。

```
user nuser ngroup;
```

上述配置中，nuser 用于指定执行工作进程的用户，ngroup 用于指定 nuser 用户所属的组。按照上述命令修改完成后，保存 nginx.conf 配置文件，平滑重启，如图 4-2 所示。从查询结果的第 2 行可以看到工作进程用户已成功修改为 nuser。

```
[root@localhost ~]# ps aux | grep nginx
root      1338  0.0  0.1  44728  1196 ?        Ss   09:16   0:00 nginx: master process /usr/local/nginx/sbin/nginx
nuser     1340  0.0  0.1  45176  1792 ?        S    09:16   0:00 nginx: worker process
root      1430  0.0  0.0 103312   860 pts/0    R+   09:19   0:00 grep nginx
[root@localhost ~]#
```

图 4-2　配置 Nginx 用户和组

多学一招：Nginx 的进程设计思想

Nginx 的进程设计思想如图 4-3 所示。它由一个主进程和多个工作进程组成，主进程接收客户端请求，转交给工作进程处理，从而很好地利用多核心 CPU 的计算能力。当管理员执行 reload 命令重新加载配置时，主进程会等待工作进程完成工作后再结束工作进程，然后基于新的配置重新创建工作进程，避免了工作过程中被打断的情况。由于整个过程中主进程没有停止，因此也不会发生漏掉客户端请求的情况。

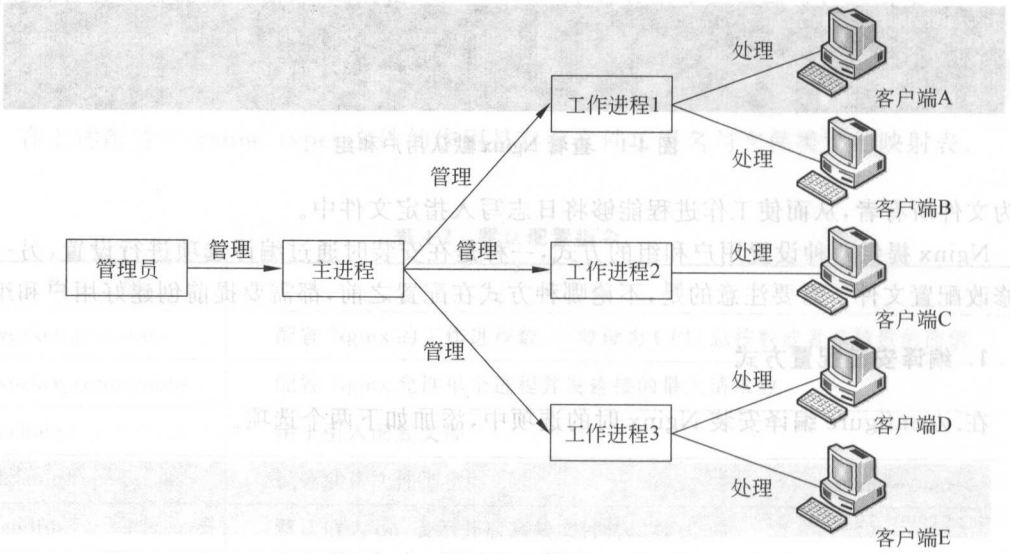


图 4-3 Nginx 进程设计思想

4.1.3 自定义错误页

在网站访问过程中，经常会遇见各种各样的错误，如找不到访问的页面则会提示 404 Not Found 错误，没有访问权限会提示 403 Forbidden 等，对于普通人而言，这样的提示界面并不友好。在 Nginx 的主配置文件中，给出了以下的处理方式。

```
error_page   500 502 503 504  /50x.html;
```

在上述配置中，error_page 指令用于自定义错误页面，500、502、503 和 504 指的就是 HTTP 错误代码，/50x.html 用于表示当发生上述指定的任意一个错误时，都使用网站根目录下的 50x.html 文件处理。

除此之外，error_page 指令还可以指定单个错误的处理页面、利用在线资源处理指定的错误，更改网站响应的状态码等多种设置，下面逐一演示自定义错误页面的几种常用使用方式。

1. 为每种类型的错误设置单独的处理方式

```
#指定网站根目录下的页面 40x.html,处理 403 错误
error_page 403   /40x.html;
#指定网站根目录下的图片 404.jpg,处理 404 错误
error_page 404   /404.jpg;
```

下面为了查看设置效果，将上述配置放到 server 块中，平滑重启 Nginx 使配置生效。接着，在浏览器中进行访问测试，当网站目录下没有指定默认索引文件时访问会发生 403 错误，如图 4-4 所示；当访问网站下不存在的目录 t 时，如图 4-5 所示。

需要注意的是，若使用 IE 浏览器运行上述示例，则自定义错误页面的大小必须大于

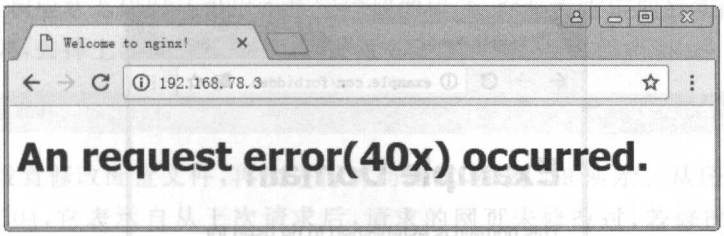


图 4-4　403 错误处理页面

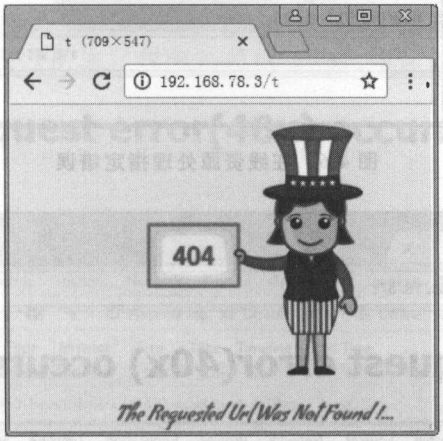


图 4-5　404 错误处理页面

512 字节，否则错误页面的展示将使用 IE 默认的错误页面。

2. 利用在线资源进行处理错误

处理错误的页面除了可以使用本站的资源外，还可以在发生指定错误时跳转到指定的 URL，利用在线资源进行处理。配置示例如下。

```
#处理单个指定错误
error_page 403      http://example.com/forbidden.html;
#处理一系列指定错误
error_page 500 502 503 504 http://example.com/notfound.html;
```

按照上述设置修改配置文件后，发生 403 错误就跳转到 http://example.com/forbidden.html 页面，测试访问效果如图 4-6 所示。

3. 更改响应状态码

在用户通过浏览器发送 HTTP 请求时，服务器处理完成后会返回响应信息，响应信息中的状态码(Status)就是服务器在处理用户 HTTP 请求后的响应状态。例如，用户访问一个不存在的页面，服务器返回的响应状态码就为 404。

利用浏览器提供的 F12 开发者工具查看到当前请求页面的状态码，如图 4-7 所示。

从图 4-7 中可以看到，当前请求地址 http://192.168.78.3/t 返回的状态码是 404。若

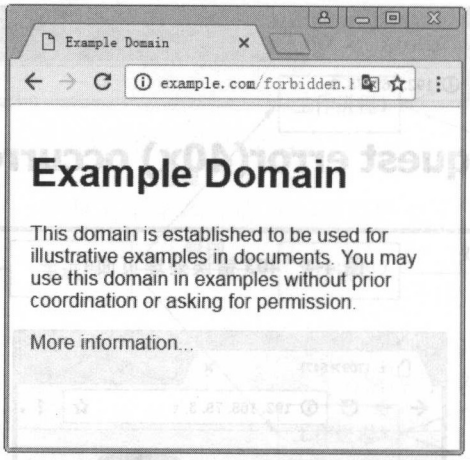


图 4-6 在线资源处理指定错误

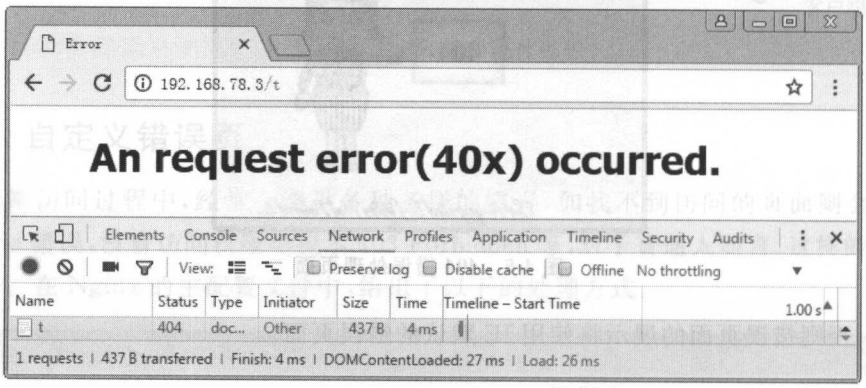


图 4-7 查看响应状态码

要隐藏服务器返回的真实状态码信息，则可以利用＝进行自定义设置，具体配置如下。

```
error_page 404 =200 /40x.html;
```

按照上述设置修改配置文件，再次进行访问测试，如图 4-8 所示。从图中可以看出，在发生 404 错误时，响应信息中的状态码是自定义的码值 200，成功隐藏了实际响应状态码。

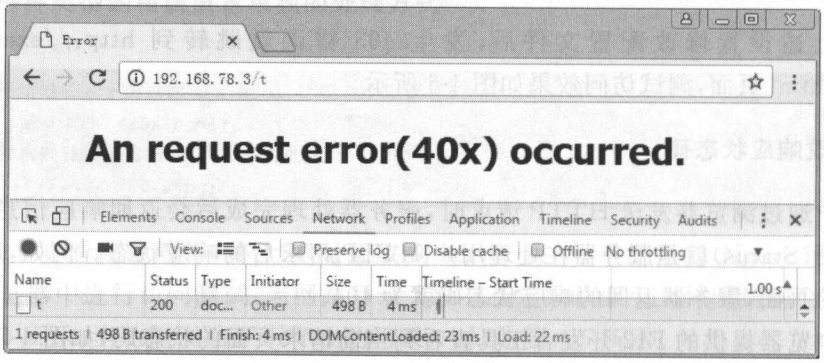


图 4-8 修改后的响应状态码

另外，更改响应状态码时还可以不指定确切的码值，而是由重定向后实际处理的真实结果来决定。例如，去掉上面配置的 200 后，配置如下。

```
error_page 404=/40x.html;
```

按照上述设置修改配置文件，再次进行访问测试，如图 4-9 所示。从图中可以看出，当前的状态码为 304，它表示自从上次请求后，请求的网页未修改过，若修改页面 40x. html 后，再次访问则会出现图 4-8 所示的效果。

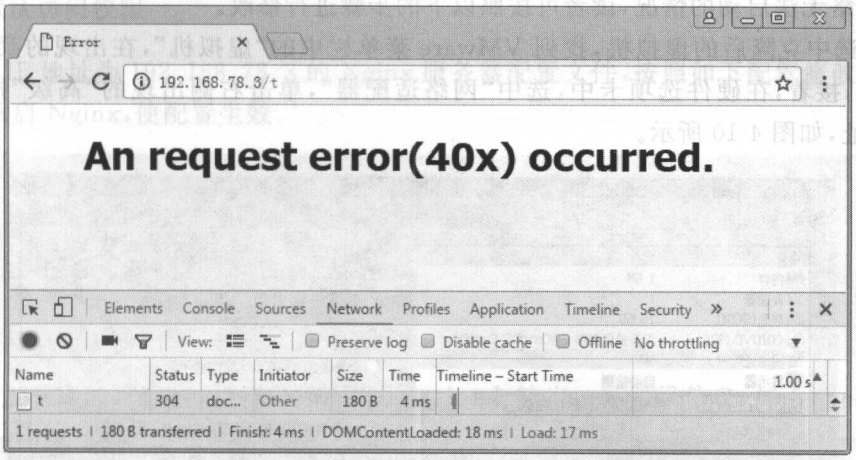


图 4-9　不明确指定响应状态码

4.2　访问控制

访问控制是网络安全防范和保护的主要策略，其任务是保证网络资源不被非法访问。Nginx 作为 Web 服务器的后起之秀，也提供了访问控制的功能。它可以根据实际需求，对用户可以访问和禁止的目录进行限制。下面将对 Nginx 提供的权限控制指令以及典型的应用进行详细讲解。

4.2.1　权限控制指令

Nginx 中提供了两个用于配置访问权限控制的指令，分别为 allow 和 deny。从其名称就可以看出，allow 用于设置允许访问的权限、deny 用于设置禁止访问的权限。在使用时，权限指令后只需跟上允许或禁止的 IP、IP 段或 all 即可。其中，all 表示所有的。

权限控制指令的使用虽然简单，但是在设置的过程中，还需要特别注意以下几个点。

- 单个 IP 指定作用范围最小，all 指定作用范围最大。
- 同一块下，若同时存在多个权限指令(deny、allow)，则先出现的访问权限设置生效，并且会对后出现的设置进行覆盖，未覆盖的范围依然生效，否则以先出现的设置为准。
- 当多个块(如 http、server、location)中都出现了权限设置指令，则内层块中的权限级别要比外层块中设置的权限级别高。

为了读者更好地理解权限控制指令，下面对几种常用使用方式分别演示说明。

1. 准备工作

由于权限的控制测试，需要涉及不同的 IP。因此，在具体讲解前，准备多台虚拟机，一台作为 Nginx 服务器，如 192.168.78.3。另外几台作为客户端，如 192.168.78.128 和 192.168.78.200，通过 curl 访问服务器(192.168.78.3)，进行测试。

当前 CentOS 系统采用了最小安装方式，在使用第 1 章讲解的克隆方式准备虚拟机时，会出现网络无法启动的情况，读者可按照以下的步骤进行修改。

(1) 选中克隆后的虚拟机，找到 VMware 菜单栏中的“虚拟机”，在出现的菜单中选择“设置”项，接着，在硬件选项卡中，选中“网络适配器”，单击右侧出现的“高级”按钮，获取 MAC 地址，如图 4-10 所示。

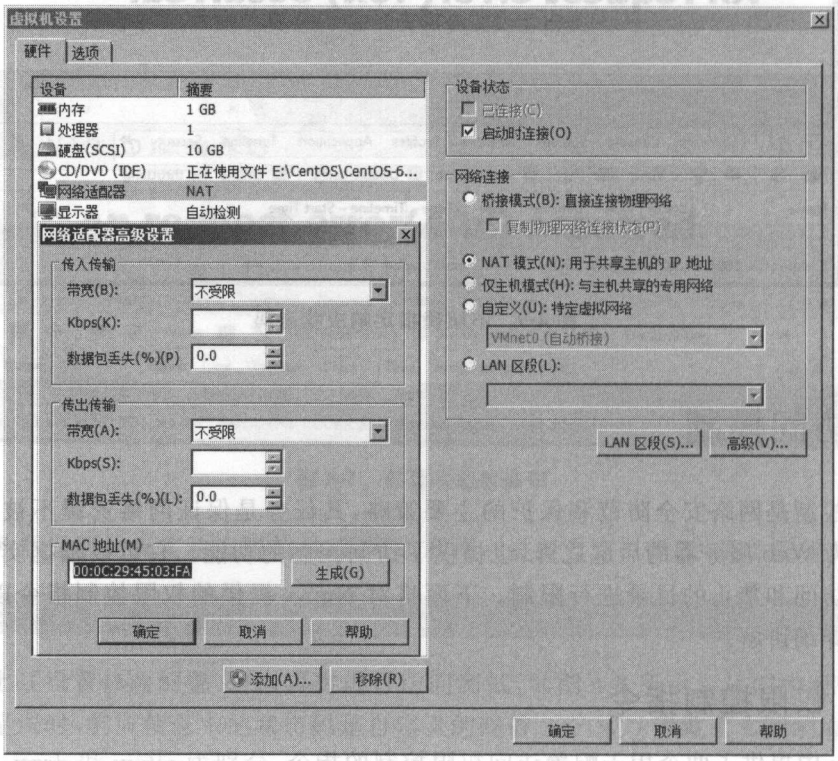


图 4-10 获取 MAC 地址

(2) 编辑克隆后的虚拟机网络配置文件/etc/sysconfig/network-scripts/ifcfg-eth0，修改 MAC 地址和静态 IP 地址，具体如下。

```
HWADDR=00:0C:29:45:03:FA
IPADDR=192.168.78.128
```

上述配置中的 HWADDR 的值与图 4-10 查询到的地址相同，IPADDR 设置为未被占用的地址即可。

(3) 重新获取克隆后虚拟机的真实信息。

```
[root@localhost ~]#start_udev
Starting udev:                [ OK ]
```

上述 start_udev 命令，可以重新启动 udev 服务，重新读取网卡设备的配置文件，并根据 ONBOOT=yes 的配置启用网卡。完成上述操作后，通过 ifconfig 命令查看 eth0 网卡是否已经正常启动。

2. 默认访问权限

打开 IP 地址为 192.168.78.3 的 Nginx 服务器配置文件，按照如下配置修改 server 块后，平滑重启 Nginx，使配置生效。

```
1    server {
2        listen       80;
3        server_name  localhost;
4        root   html;
5        index  index.html index.htm;
6    }
```

为了方便测试，在 html 目录中编写 index.html，输出一行提示信息，如下所示。

```
<h1>Welcome 192.168.78.3!</h1>
```

接着，利用之前准备的两个客户端(192.168.78.128 和 192.168.78.200)，通过 curl 请求服务器地址 192.168.78.3，具体命令如下。

```
curl http://192.168.78.3
```

执行完上述命令后，若在两个客户端中都能看到如图 4-11 所示的结果，表明虚拟主机默认未设置访问权限时，允许所有用户的访问，效果相当于为 server 设置 allow all。

3. 禁止所有用户的访问

在步骤 2 中第 5 行下添加以下指令，用于禁止所有的客户端访问，具体如下。

```
deny all;
```

接着，再次使用两个客户端进行访问测试，访问结果如图 4-12 所示。从图中可以看出，页面显示 403 Forbidden，表明禁止访问成功。

```
[root@localhost ~]# curl http://192.168.78.3

<h1>Welcome 192.168.78.3!</h1>

[root@localhost ~]#
```

图 4-11　默认访问权限设置

```
[root@localhost ~]# curl http://192.168.78.3
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.10.1</center>
</body>
</html>
[root@localhost ~]#
```

图 4-12　禁止所有客户端访问

需要注意的是，在server块下设置deny all后，服务器(192.168.78.3)内的客户端软件在访问自己时也会出现403 Forbidden。因此，读者在设置时需要慎重考虑。

4. 只允许指定用户访问

对于服务器192.468.78.3来说，若仅允许客户端192.168.78.128访问，则将第3步中的权限设置修改成以下配置即可。

```
allow 192.168.78.128;
deny all;
```

上述指令表示只允许192.168.78.128客户端访问，其他所有客户端都不能访问。利用之前准备的两个客户端可以进行测试。需要注意的是，若省略此处的deny all，则会允许所有客户端访问；若将deny all移动到allow 192.168.78.128之后，则会阻止所有客户端访问。

从上述规律看出，同一个块下的两个权限指令，先出现的设置会覆盖后出现的设置，使得allow 192.168.78.128的配置优先生效；同时deny指令设置的访问范围all较大，未被allow覆盖的范围配置依然生效，达到除了IP为192.168.78.128的用户外，禁止其他用户对服务器访问的效果。

5. 不同块间的权限指令优先级

为了测试不同块间的权限指令优先级，重新配置服务器192.168.78.3的权限设置，在http块中设置禁止所有用户对http块的访问，具体配置如下。

```
1   http {
2       ...
3       deny all;
4       server {
5           listen       80;
6           server_name  localhost;
7           root   html;
8           index  index.html index.htm;
9       }
10  }
```

接着，使用之前准备的两个客户端访问测试，可以看到如图4-12所示的访问失败结果。然后，在第8行指令的配置后添加以下配置。

```
allow all;
```

修改完成后，使用两个客户端访问，可以看到如图4-11所示的访问成功结果。这是由于Nginx配置文件中的各个块在嵌套的情况下，内层块内的指令比外层块内的指令执行优先级高。因此，当内外层块中同时出现权限指令时，则内层块中的allow all会覆盖外层块中的deny all的设置。

4.2.2　访问控制典型应用

在实际应用中，权限控制的需求更加复杂。例如，对于网站下的 img 目录允许所有用户访问，但对于网站下的 admin 目录则仅允许管理员身份的用户访问。此时，仅靠 deny 和 allow 这两个权限指令不能满足用户的需求，还需要使用 location 块来完成相关需求的匹配。

在此之前，首先要简单了解一下 location 的相关语法及规定，具体如下。

```
location  [=|~|~*|^~]  URI  { … }          #语法类型 1
location  @name  { … }                     #语法类型 2
```

在上述语法中，=、～、～*、^～和@都是 location 用于实现访问控制的前缀，且在使用时只能选择一种，当然也可以不设置前缀。其中，关于 location 前缀的含义如表 4-3 所示。URI 表示 URL 地址中从域名到参数之间的部分，{ … }表示指令块，用于满足 location 匹配条件后需要执行的指令。

表 4-3　location 的前缀

前缀	说　明
=	根据其后的指定模式进行精准匹配。例如，在访问时要与/html/aaa/index.html 完全一致才会执行其后的指令块
～	使用正则表达式完成 location 的匹配，区分大小写
～*	使用正则表达式完成 location 的匹配，不区分大小写
^～	不使用正则表达式，完成以指定模式开头的 location 匹配
@	用于定义一个 location 块，且该块不能被外部客户端所访问，只能被 Nginx 内部配置指令所访问

根据表 4-3 的描述，可将 location 根据不同前缀的使用方式，大致分为普通 location 和正则 location。其中，～和～* 属于正则 location，其余的前缀和没有前缀的情况都属于普通 location。

接下来，通过 location 块和权限控制指令，逐一演示访问控制的几种典型使用方式。

1. 精准匹配

所谓精准匹配指的就是用户访问的 URI 与指定的 URI 完全一致的情况，才会执行其后的指令块，示例配置如下。

```
1   server {
2       listen       80;
3       server_name  localhost;
4       root html;
5       index index.html index.htm;
6       location = /js {
7           allow 192.168.78.128;
8       }
9       location = /admin/auth {
```

```
10          allow 192.168.78.200;
11      }
12      deny all;
13  }
```

上述第6～8行和第9～11行配置设置了两个精准匹配，第12行用于禁止所有用户的访问。当上述配置中允许访问的两个客户端，请求网站根目录下不存在的文件或目录时，如果符合匹配规则，网页显示404 Not Found，不符合时显示403 Forbidden。

假设网站根目录下没有任何文件，下面使用IP为192.168.78.128的A用户和IP为192.168.78.200的B用户通过不同的URL进行访问测试，其对应的响应结果如表4-4所示。

表4-4 精准匹配测试

URL	A用户响应结果	B用户响应结果
http://192.168.78.3	403 Forbidden	403 Forbidden
http://192.168.78.3/js	404 Not Found	403 Forbidden
http://192.168.78.3/admin/auth	403 Forbidden	404 Not Found
http://192.168.78.3/admin	403 Forbidden	403 Forbidden

从表4-4可以看出，精准匹配是只有用户请求的URI与location中定义的匹配模式完全一致的情况下，才会执行其后的指令块，否则匹配不成功。

2. 正则匹配

Nginx配置文件中，多个正则location之间按照正则location在配置文件中的书写顺序进行匹配，且只要匹配成功就不会继续匹配后面定义的正则location。下面在IP为192.168.78.3的虚拟机中，设置以下两个正则location访问控制，具体如下。

```
1   location ~\.html$ {
2       allow all;
3   }
4   location ~^/aaa/.*\.html$ {
5       deny all;
6   }
```

在上述配置中，第1行表示匹配网站根目录下以.html结尾的文件，第4行表示匹配网站根目录下aaa目录中以.html结尾的文件。

下面使用IP为192.168.78.128的用户通过不同的URL进行访问测试，其对应的响应结果如表4-5所示。

表4-5 正则匹配测试

URL	响应结果
http://192.168.78.3/test.html	404 Not Found(匹配了第1行location)
http://192.168.78.3/aaa/test.html	404 Not Found(匹配了第1行location)

从表 4-5 可以看出，当 location 中的 URI 与用户请求中以. html 为结尾的文件匹配上时，正则 location 停止了继续匹配，因此显示结果都为 404 Not Found。

接下来，调换第 1～3 行与第 4～6 行代码的编写顺序，再次访问 http://192.168.78.3/test.html 结果依然为 404 Not Found，而访问 http://192.168.78.3/aaa/test.html 的结果为 403 Forbidden。

从上述两组测试对比可总结出，正则 location 的编写顺序不同，则结果不同，且只有前面定义的正则 location 匹配不成功的情况下，才会继续匹配后面的正则 location。因此，读者在实际应用中要注意正则 location 在配置文件中的书写顺序。

3. 最大前缀匹配

由于 location 可以同时定义多个，当一个配置文件中同时出现多个 location 时，普通 location 之间遵循“最大前缀匹配”原则。通俗地讲就是，匹配度最高的 location 将会执行，示例如下。

```
1    location /ng.test {
2        allow all;
3    }
4    location /ng.test/log {
5        deny all;
6    }
```

为了方便对比学习，下面利用不同的 URL 进行访问测试，对应的响应结果如表 4-6 所示。

表 4-6　最大前缀匹配

URL	响应结果
http://192.168.78.3/ng.test/data	404 Not Found(匹配了第 1 行 location)
http://192.168.78.3/ng.test/log	403 Forbidden(匹配了第 4 行 location)
http://192.168.78.3/ng.test/log/date	403 Forbidden(匹配了第 4 行 location)

值得一提的是，当最大前缀 location 与正则 location 同时存在时，如果正则 location 匹配成功，则不会执行最大前缀 location。具体示例如下。

```
1    location / {
2        deny all;
3    }
4    location ~\.html$ {
5        allow all;
6    }
```

上述配置中，第 1～3 行定义的是最大前缀 location，用于匹配当前网站根目录下的所有文件，第 4～6 行用于正则匹配所有以. html 结尾的 URI。不同 URL 及其对应的响应结果如表 4-7 所示。

表 4-7 普通与正则 location

URL	响应结果
http://192.168.78.3	403 Forbidden(匹配了第 1 行 location)
http://192.168.78.3/notfound.html	404 Not Found(匹配了第 4 行 location)
http://192.168.78.3/index.php	403 Forbidden(匹配了第 1 行 location)

从表 4-7 中可以看出，当用户访问 http://192.168.78.3 时，完成第 1 行的匹配；而在用户访问 http://192.168.78.3/notfound.html 和 http://192.168.78.3/index.php 时，前者符合正则 location，结果为 404 Not Found，而后者不符合正则 location，显示了最大前缀匹配的结果 403 Forbidden。

脚下留心：location / {}与 location =/ {}的区别

location / {}遵守普通 location 的最大前缀匹配，由于任何 URI 都必然以“/”根开头，所以对于一个 URI，若配置文件中有更合适的匹配则会将其替代，否则返回 location / {}匹配到的结果，它相当于站点默认配置。

而 location=/ {}遵守的是精准匹配，也就是只能匹配该站点根目录，同时会禁止继续搜索正则 location，效率比 location / {}要高。因此，若在开发中能确定精准匹配的情况，可以采用 location =/ {}的方式，提升匹配效率。

4. 禁用正则匹配

利用=精准匹配或^～非正则匹配可以在正则匹配之前优先匹配，从而禁止执行原有的正则匹配。下面在 server 块中添加以下几条 location 匹配规则，具体如下。

```
1  location =/aaa/test.html {
2      allow all;
3  }
4  location ^~/ {
5      deny all;
6  }
7  location ~\.html$ {
8      allow all;
9  }
```

在上述配置中，第 1 行仅用于精准匹配网站根目录下的/aaa/test.html，第 4 行用于非正则匹配网站根目录下的文件，第 7 行用于正则匹配网站根目录下以.html 为结尾的文件。

接下来通过不同 URL 进行访问测试，具体如表 4-8 所示。从表中的响应结果可以看出，在使用了“=”或“^～”前缀时，普通 location 匹配后将不再执行正则 location 的匹配。

值得一提的是，前缀“=”和“^～”虽然都能阻止继续搜索正则 location，不同的地方是它们遵循的规则不同，“^～”依然遵循最大前缀匹配规则，而=则严格按照精准匹配执行。

因此，当多种类型的 location 匹配同时出现时，最终执行结果为“=”匹配优先于“^～”匹配，“^～”匹配优先于正则匹配，正则匹配优先于普通的最大前缀匹配。只要优先的 location 匹配成功，就不会执行其他的 location。

表 4-8　禁用正则匹配测试

URL	响应结果
http://192.168.78.3	403 Forbidden(匹配了第 4 行 location)
http://192.168.78.3/index.html	403 Forbidden(匹配了第 4 行 location)
http://192.168.78.3/bbb/test.html	403 Forbidden(匹配了第 4 行 location)
http://192.168.78.3/aaa/test.html	404 Not Found(匹配了第 1 行 location)

多学一招：root 与 alias 的区别

在 location 中指定目录时，除了可以使用 root 指令外，还可以使用 alias 指令完成。两者在使用时有一定的区别，具体示例如下。

```
#当收到"/img/itheima.png"请求时,将请求映射为"/var/www/image/itheima.png"
location /img/ {
    alias /var/www/image/;
}
#当收到"/img/itheima.png"请求时,将请求映射为"/var/www/image/img/itheima.png"
location /img/ {
    root /var/www/image;
}
```

从上述示例可以看出，alias 在映射路径时不会追加 location 匹配到的部分，而 root 追加了 location 匹配到的部分。

4.3　日志文件

日志的有效使用有利于项目开发与维护的统计排错。Nginx 提供了一个非常灵活的日志记录功能，它可以使每个块的配置拥有各自独立的日志进行记录，并且根据记录内容的不同又分为访问日志和错误日志。接下来本节将对 Nginx 日志功能的使用进行详细介绍。

4.3.1　访问日志

访问日志主要用于记录客户端访问 Nginx 的每一个请求，格式可通过 log_format 指令进行自定义，存储路径、缓存大小等可使用 access_log 指令设置。通过访问日志的配置，可以记录用户 IP、访问时间、请求方式、响应状态、地域来源、跳转来源、使用终端等信息。

1. 查看默认访问配置

打开 Nginx 的配置文件 nginx.conf，找到 log_format 与 access_log 指令的默认配置，具体如下。

```
1   log_format  main  '$remote_addr-$remote_user [$time_local] "$request" '
2                     '$status $body_bytes_sent "$http_referer" '
3                     '"$http_user_agent" "$http_x_forwarded_for"';
4   access_log  logs/access.log  main;
```

上述第 1～3 行用于设置访问日志的格式，main 表示访问日志格式名称，用户可以自定义。其后的字符串表示访问日志格式样式。其中相关的内置变量的含义（如 $remote_addr）参考表 4-9 所示。

第 4 行中 access_log 指令的第 1 个参数 logs/access.log 用于指定相对于 Nginx 的安装目录/usr/local/nginx 的日志文件存放路径，并包含日志文件名称，第 2 个参数表示由 log_format 指令定义的日志格式名称。

表 4-9 与日志格式相关的内置变量

内置变量	含义
$remote_addr	客户端的 IP 地址
$remote_user	客户端用户名，用于记录浏览者进行身份验证时提供的名称，如果没有登录则为空
$time_local	访问的时间与时区，如 21/Sep/2016:12:21:25 +0800，时间信息最后的 +0800 表示服务器所处时区位于 UTC 之后的 8 小时
$request	请求的 URI 和 HTTP 协议，如 GET / HTTP/1.1
$status	记录请求返回的 HTTP 状态码，如 200（成功）
$body_bytes_sent	发送给客户端的文件主体内容的大小，如 899
$http_referer	来路 URL 地址
$http_user_agent	客户端浏览器信息
$http_x_forwarded_for	客户端 IP 地址列表（包括中间经过的代理）

需要注意的是，Nginx 默认开启了访问日志的功能，且 log_format 指令的配置仅可用在 http 块内，否则会出现警告信息。

为了查看访问日志文件记录的内容，在浏览器端多次访问 Nginx 默认的网站，如 http://192.168.78.3，然后切换到日志存放的目录/usr/local/nginx/logs/中，打开文件 access.log 查看，记录内容如图 4-13 所示。

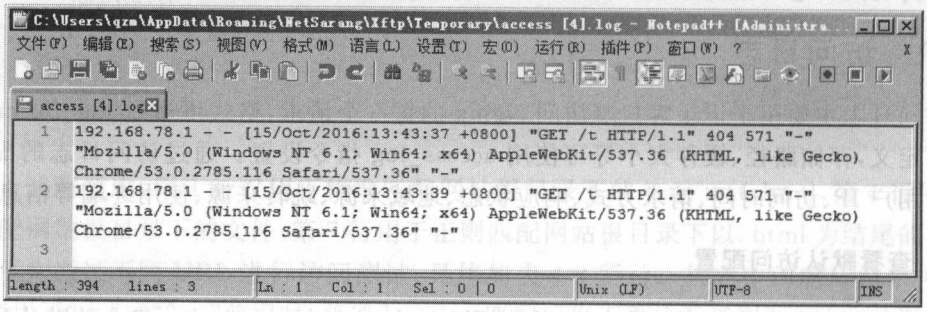


图 4-13 查看默认日志文件

2. 自定义日志格式

修改 http 块中的默认设置，自定义访问日志格式 mylog，具体如下。

```
log_format  mylog  '[ip:] $remote_addr  [time:] $time_local  [user_agent:] "$http_user_
agent"';
```

上述访问日志格式的名称自定义为 mylog，并且修改了默认访问日志的存储格式，在每个需要记录的数据前面都使用“[名称:]”的形式进行标注。

接着，在 Nginx 的默认 server 块中采用 mylog 访问日志格式，使用 access_log 指令将其访问日志文件保存到指定的目录中，具体如下。

```
access_log  logs/192.168.78.3/access.log  mylog buffer=2k flush=5s;
```

在上述配置中，buffer 参数用于设置内存缓存区的大小，flush 参数用于设置内容保存在缓存区中的最大时间。在 logs 目录下手动创建 192.168.78.3 目录，同时要保证当前 Nginx 进程的用户和组有对该目录创建 access.log 文件的权限。否则，日志文件将无法被创建。

完成设置后，平滑重启 Nginx，可以看到生成的日志文件，如图 4-14 所示。

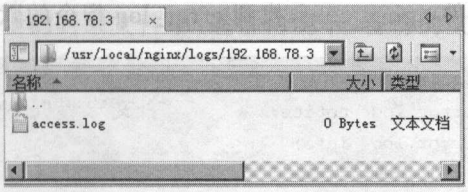


图 4-14　查看自定义生成的日志文件

利用浏览器访问测试后，日志文件的查看结果如图 4-15 所示。从图中可以看出，当前已经按照自定义的格式进行日志记录。

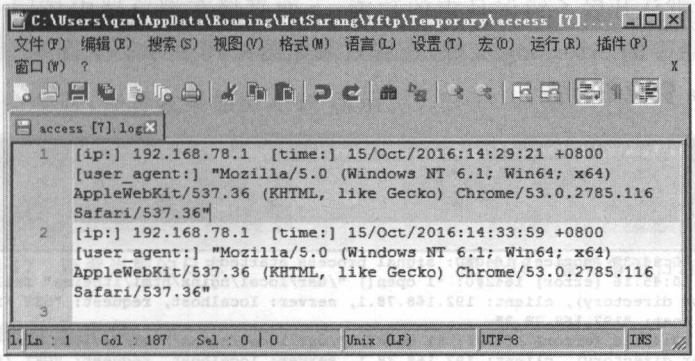


图 4-15　查看自定义日志文件格式

值得一提的是，若在访问过程中需要记录子请求的日志记录，则可以将 log_subrequest 指令设置为 on，否则默认不记录。

3. 关闭访问日志

关闭访问日志的实现非常简单，只需要将 access_log 指令的参数设置为 off 即可。接下来，在上述配置访问日志的 server 块中添加以下设置，关闭访问日志。

```
access_log off;
```

接着，平滑重启 Nginx，多次访问 192.168.78.3，查看其访问日志文件 access.log 的记录，如果没有增加新的记录，表明当前网站的访问将不再被记录。

由此可见，在使用日志记录访问痕迹、分析排错时，可以结合 location 指令与 access_log off 关闭不必要的请求记录，避免对分析排错的过程造成麻烦。例如，网站上线后，一些搜索引擎的蜘蛛、爬虫会产生大量的访问痕迹。

4.3.2 错误日志

错误日志是由 error_log 指令设置的，主要是用来记录客户端在访问 Nginx 时出错的记录，且该错误显示格式不支持自定义功能。通过错误日志，可以查看到系统某个服务的性能瓶颈等。有效地利用错误日志，可以得到很多有价值的信息。

1. 默认错误日志配置

打开 Nginx 的配置文件 nginx.conf，找到 error_log 指令的默认配置，具体如下。

```
1    error_log  logs/error.log;
2    error_log  logs/error.log  notice;
3    error_log  logs/error.log  info;
```

在 Nginx 给出的以上 3 种配置方式中，error_log 指令的第 1 个参数用于存放错误日志的路径，第 2 个参数用于指定错误记录详细程度的等级，默认值为 error，如第 1 行配置就是 error 等级的设置。其他常用的等级还有 debug、info、notice、warn、error 和 crit，日志记录详细程度依次递减，debug 记录的内容最详细，crit 记录的内容最简洁。

由于 Nginx 默认开启了错误日志的功能，下面直接在浏览器中访问一个不存在的文件，打开 logs 目录下的 error.log 文件，查看效果如图 4-16 所示。从图 4-16 中可以看出，用户访问的 itheima 资源不存在。

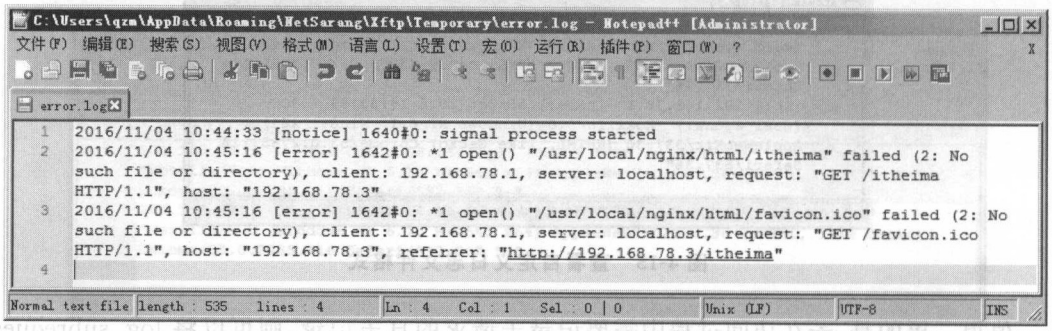


图 4-16 查看默认错误日志记录

在实际应用中，error_log 指令除了可以像默认配置一样在 main 块中设置，还可以在 http、server、location 块中设置，配置方式相同。

2. 关闭错误日志

Nginx 关闭错误日志的实现方式比较特殊，具体设置如下。

```
error_log /dev/null;
```

上述配置的含义就是，将错误日志信息全部输出到 Linux 的空设备中，表示丢掉输出信息，也可以将其当做一个“垃圾桶”。

4.3.3　日志文件切割

为了使日志文件的存储更合理、有序，可以通过切割的方式将 Nginx 中的日志文件按照规定的时间分开存储。其中，切割的方式可分为手动切割和自动切割两种。

1. 手动切割

所谓手动切割方式就是用户自己执行相关的命令，共分为两步，第一步使用 mv 命令将需要备份的日志移到一个新的目录文件中，通常情况下，备份的文件使用日期命名。第二步重新生成一个空的日志文件，以便存储新的记录。具体操作步骤如下。

(1) 查看当前 Nginx 中的日志文件。

```
[root@localhost ~]#cd /usr/local/nginx/logs
[root@localhost logs]#ls
```

执行完上述命令后，如图 4-17 所示。

(2) 备份 192.168.78.3 目录下的日志文件，以“年月日”作为文件名称进行备份。

```
[root@localhost logs]#cd 192.168.78.3
[root@localhost 192.168.78.3]#mv access.log 20160825.log
```

执行完上述命令后，查看效果如图 4-18 所示。

```
[root@localhost ~]# cd /usr/local/nginx/logs
[root@localhost logs]# ls
192.168.78.3  error.log  nginx.pid
[root@localhost logs]#
```

图 4-17　查看当前的日志

```
[root@localhost 192.168.78.3]# ls
20160825.log  error.log
[root@localhost 192.168.78.3]#
```

图 4-18　查看备份日志文件

(3) 生成新的日志文件。

接着，利用 Nginx 的 reopen 功能，完成新日志文件的生成。

```
[root@localhost 192.168.78.3]#nginx -s reopen
```

执行完上述命令后的效果如图 4-19 所示。

```
[root@localhost 192.168.78.3]# ls
20160825.log  access.log  error.log
[root@localhost 192.168.78.3]#
```

图 4-19　查看新的日志文件

2. 自动切割

手动切割固然实现简单，但若是每天都需要进行相关的操作未免浪费时间，因此可以编写一个 shell 脚本文件，使用 Linux 系统提供的

crontab 指令让其每天按照设定的时间自动备份前一天的日志。下面为 Nginx 默认 server 块中的日志文件设置自动切割。

1）创建脚本文件 autolog.sh

```
1  #!/bin/bash
2  #当前 Nginx 日志文件存放的目录
3  logs_path="/usr/local/nginx/logs/192.168.78.3"
4  #备份日志文件
5  mv $logs_path/access.log $logs_path/`date +"%Y%m%d%H%M"`.log
6  #重新打开 Nginx 日志
7  /usr/local/nginx/sbin/nginx -s reopen
```

上述第 5 行代码中，access.log 指的是/usr/local/nginx/logs/192.168.78.3 目录下的一个日志文件名称。而 date 表示获取的当前时间，%Y%m%d%H%M 表示以“年月日时分”的形式表示前面获取到的时间。第 7 行代码用于重新打开 Nginx 日志文件。

接着，为 autolog.sh 脚本文件设置可执行权限，具体如下。

```
chmod +x autolog.sh
```

另外，若要以“年月日”的形式保存前一天的日志，可将第 5 行代码修改成如下形式。

```
mv $logs_path/access.log $logs_path/`date -d yesterday +"%Y%m%d"`.log
```

上述代码中的 date -d yesterday 表示执行该文件时的前一天。

2）系统自动备份

要实现每天定点系统自动备份日志文件，可以使用 Linux 中提供的 crontab 命令，设置某脚本被周期性执行的任务。执行下列命令后，进入任务的编辑页面。

```
[root@localhost 192.168.78.3]#crontab -e
```

上述命令中的-e 表示编辑当前用户的定时任务。进入编辑页面后的操作与 vi 编辑器的操作相同，在实际应用中，通常按照如下形式添加命令，设置系统自动备份时间。

```
0 0 * * * /usr/local/nginx/logs/192.168.78.3/autolog.sh >/dev/null 2>&1
```

上述命令用于每天晚上零点备份一次日志文件，关于上述命令参数的含义如表 4-10 所示。其中，* 表示“每”，如第 1 个参数设置为 * 表示每分钟执行一次后面的命令，依次类推。

表 4-10 crontab -e 编辑页面的相关参数

参　数	说　明
第 1 个参数	表示分钟，取值范围为 0～59，0 表示整点
第 2 个参数	表示小时，取值范围为 0～23，0 表示子夜
第 3 个参数	表示日，取值范围为 1～31
第 4 个参数	表示月，取值范围为 1～12
第 5 个参数	表示星期，取值范围为 0～6，其中 0 表示星期天，1 表示星期一，以此类推
第 6 个参数	表示要运行的命令，如这里使用 bash 运行编写好的 autolog.sh 脚本文件

值得一提的是，>/dev/null 2>&1 指令用于屏蔽标准输出和标准出错的信息，并将其放到“垃圾桶”中，目的就是防止在 Linux 系统中执行 crontab 操作时，将输出内容和错误信息以邮件的形式发送个用户，造成大量的垃圾文件，因此，在使用 crontab 指令时，推荐添加 >/dev/null 2>&1 设置。

为了方便测试，这里将 crontab 任务编辑页面的时间修改成如下形式。

```
* * * * * /usr/local/nginx/logs/192.168.78.3/autolog.sh >/dev/null 2>&1
```

修改完成后，crontab 程序备份日志的频率将为每分钟备份一次。任务编辑完成后，可以通过“crontab -l”显示当前用户的 crontab 文件内容，“crontab -r”指令删除设置的任务计划。

接下来，在完成上述设置的几分钟后，查看日志文件自动切割效果，如图 4-20 所示。

```
[root@localhost 192.168.78.3]# ll
total 12
-rw-r--r--. 1 root  root  748 Oct 15 15:02 20160825.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:04 201610151711.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:11 201610151712.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:12 201610151713.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:13 201610151714.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:14 201610151715.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:15 201610151716.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:16 201610151717.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:17 201610151718.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:18 201610151719.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:19 201610151720.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:20 201610151721.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:21 201610151722.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:22 201610151723.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:23 201610151724.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:24 201610151725.log
-rw-r--r--. 1 nuser root    0 Oct 15 17:25 access.log
-rwxr-xr-x. 1 root  root  263 Oct 15 16:49 autolog.sh
-rw-r--r--. 1 nuser root 3011 Oct 15 16:34 error.log
[root@localhost 192.168.78.3]#
```

图 4-20　查看日志文件自动切割效果

4.4　虚拟主机

4.4.1　什么是虚拟主机

虚拟主机技术是指在一台物理主机服务器上划分出多个磁盘空间，每个磁盘空间都是一台虚拟主机，每台虚拟主机都可以独立对外提供 Web 服务，且互不干扰。在外界看来，虚拟主机就是一台独立的服务器主机，这就意味着用户能够利用虚拟主机把多个不同域名的网站部署在同一台服务器上，而不必再为建立一个网站单独购买一台服务器，既解决了维护服务器技术的难题，同时又极大地节省了服务器硬件成本和相关的维护费用。

例如，在一台物理主机服务器(10.20.30.40)上划分出多台虚拟主机，同时在每台虚拟主机上部署并运行一个网站，当用户请求时，物理主机服务器根据其配置情况，将用户的请求分配到不同的虚拟主机上进行处理，如图 4-21 所示。

4.4.2　基于端口号配置虚拟主机

基于端口号配置虚拟主机的方式，是 Nginx 中配置虚拟主机最简单的方式。它的原理

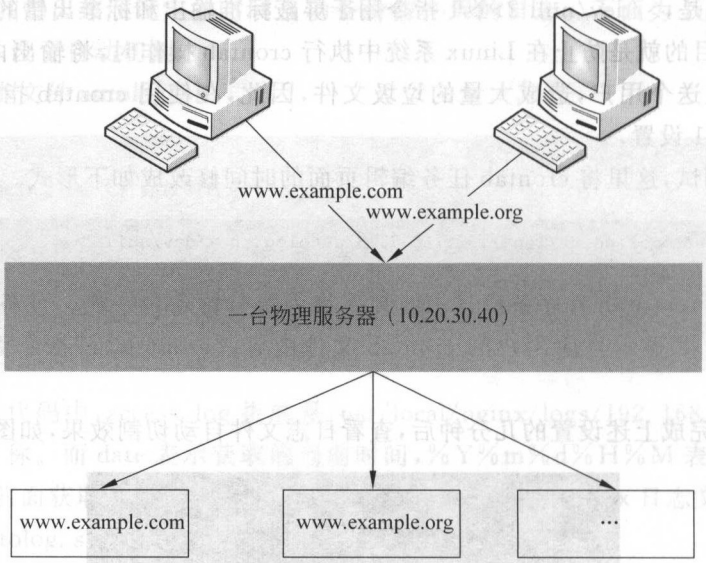


图 4-21 虚拟主机示意图

就是一个 Nginx 监听多个端口，根据不同的端口号，来区分不同的网站。

假设当前物理主机的 IP 为 192.168.78.3，然后让其分别监听不同的端口，如 8001 和 8002，来实现根据不同端口号配置虚拟主机的功能。

(1) 在配置虚拟主机前，首先打开 Nginx 的配置文件 nginx.conf，查看默认配置文件中提供的关于虚拟主机配置的方法，具体如下。

```
1   #another virtual host using mix of IP-, name-, and port-based configuration
2   #server {
3   #    listen       8000;
4   #    listen       somename:8080;
5   #    server_name  somename  alias  another.alias;
6   #    location / {
7   #        root   html;
8   #        index  index.html index.htm;
9   #    }
10  #}
```

上述第 1 行注释，用于告知用户 Nginx 中虚拟主机的配置可以基于 IP 地址、域名和端口号进行设置，第 2～10 行配置是用于在 server 块中完成虚拟主机的设置。其中，第 3 行配置表示使用 listen 命令监听端口，第 4 行配置表示使用“IP/域名:端口号”的方式监听端口，在实际设置中两者只能选其一。

因此，若要在 Nginx 中配置一个虚拟主机，只需在 http 块中添加一个 server 块即可。换句话说，http 块中的每个 server 块都是一个虚拟主机。

(2) 修改 nginx.conf 配置文件，在 http 块中添加以下两个 server 配置。

```
1   #配置监听 8001 端口号的虚拟主机
```

```
    server {
        listen       8001;
        server_name  192.168.78.3;
        root    html/html8001;
        index   index.html index.htm;
    }
    #配置监听 8002 端口号的虚拟主机
        server {
        listen       8002;
        server_name  192.168.78.3;
        root    html/html8002;
        index   index.html index.htm;
    }
```

在上述配置中，监听 8001 端口的网站根目录设置为“html/html8001”，监听 8002 端口的网站根目录设置为“html/html8002”。完成上述配置后，保存 nginx.conf 文件，平滑重启 Nginx 使设置生效。

(3) 编写测试文件并查看结果。

首先，在/usr/local/nginx/html/下分别创建目录 html8001 和 html8002。然后，在不同网站根目录下放置一个测试文件用于访问测试。

① 在 html8001 目录中创建 index.html 测试文件，编写内容如下。

```
<h1>Welcome 192.168.78.3:8001!</h1>
```

② 在 html8002 目录中创建 index.html 测试文件，编写内容如下。

```
<h1>Welcome 192.168.78.3:8002!</h1>
```

最后，在浏览器中分别访问 http://192.168.78.3:8001 和 http://192.168.78.3:8002。测试成功后效果分别如图 4-22 和图 4-23 所示。需要注意的是，若不能访问，需要修改防火墙开放 8001 和 8002 端口。

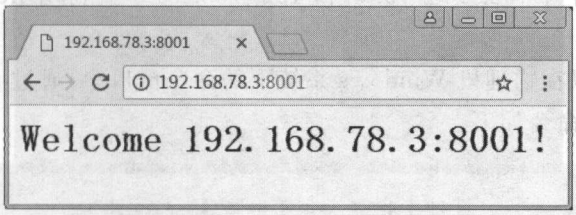


图 4-22　端口号为 8001 的虚拟主机

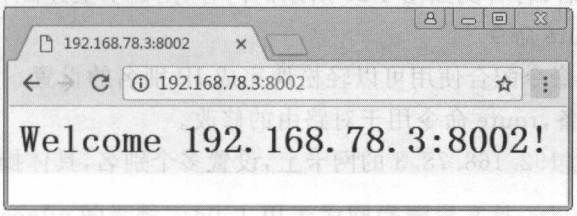


图 4-23　端口号为 8002 的虚拟主机

4.4.3 基于IP配置Nginx虚拟主机

在Linux系统中，可以通过设置IP别名的方式，实现一块物理网卡上绑定多个IP地址。接下来，将详细讲解实现基于IP配置Nginx虚拟主机的步骤。

1. 设置IP别名

在Linux上，IP别名的设置有两种方式，一种是修改网络配置文件ifcfg-eth0，一种是通过ifconfig和route命令进行设置。假设当前虚拟机的IP为192.168.78.3，下面分别对设置IP别名的两种方式进行介绍。

1）修改网络配置文件

首先，切换到网络配置文件ifcfg-eth0所在的目录，具体命令如下。

```
[root@localhost ~]#cd /etc/sysconfig/network-scripts/
```

然后，根据需要设置IP别名的个数，复制对应个数的网络配置文件ifcfg-eth0。例如，需要设置两个IP别名192.168.78.4和192.168.78.5，就将ifcfg-eth0复制为ifcfg-eth0:1和ifcfg-eth0:2两个新文件。具体操作如下。

```
[root@localhost network-scripts]#cp ifcfg-eth0 ifcfg-eth0:1
[root@localhost network-scripts]#cp ifcfg-eth0 ifcfg-eth0:2
```

接着，编辑文件ifcfg-eth0:1和ifcfg-eth0:2文件，修改DEVICE(网卡物理设备名称)和IPADDR(IP地址)，参照配置如下：

```
DEVICE=eth0:1
IPADDR=192.168.78.4
```

按照上述示例完成ifcfg-eth0:1和ifcfg-eth0:2文件的设置后，执行service network reload命令使配置生效。执行完上述命令后，就会在IP地址为192.168.78.3的网卡上添加两个IP别名，分别为192.168.78.4和192.168.78.5。利用ifconfig查看的效果如图4-24所示。

除此之外，还可以在物理机Windows系统下的命令窗口中，通过ping命令测试已设置的IP别名，具体执行命令如下。

```
ping 192.168.78.4
ping 192.168.78.5
```

执行完上述命令后，若看到如图4-25所示的内容，则证明上述配置无误。

2）ifconfig和route命令

ifconfig和route命令配合使用可以轻松地完成IP别名的设置。其中，ifconfig命令用来查看和配置网络设备，route命令用于对路由的修改。

假设在IP地址为192.168.78.3的网卡上，设置多个别名，具体操作示例如下。

```
[root@localhost ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:4A:69:E0
          inet addr:192.168.78.3  Bcast:192.168.78.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4a:69e0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:320 errors:0 dropped:0 overruns:0 frame:0
          TX packets:292 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:44811 (43.7 KiB)  TX bytes:59107 (57.7 KiB)
          Interrupt:18 Base address:0x2000

eth0:1    Link encap:Ethernet  HWaddr 00:0C:29:4A:69:E0
          inet addr:192.168.78.4  Bcast:192.168.78.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:18 Base address:0x2000

eth0:2    Link encap:Ethernet  HWaddr 00:0C:29:4A:69:E0
          inet addr:192.168.78.5  Bcast:192.168.78.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:18 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

[root@localhost ~]#
```

图 4-24　查看 IP 别名

```
C:\Users\qzm>ping 192.168.78.4

正在 Ping 192.168.78.4 具有 32 字节的数据:
来自 192.168.78.4 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.78.4 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.78.4 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.78.4 的回复: 字节=32 时间<1ms TTL=64

192.168.78.4 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\qzm>ping 192.168.78.5

正在 Ping 192.168.78.5 具有 32 字节的数据:
来自 192.168.78.5 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.78.5 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.78.5 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.78.5 的回复: 字节=32 时间<1ms TTL=64

192.168.78.5 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\qzm>_
```

图 4-25　物理机下验证 IP 别名设置

```
[root@localhost ~]# ifconfig eth0:1 192.168.78.4 broadcast 192.168.78.255 netmask 255.255.
255.0 up
[root@localhost ~]# route add -host 192.168.78.4 dev eth0:1
[root@localhost ~]# ifconfig eth0:2 192.168.78.5 broadcast 192.168.78.255 netmask 255.255.
255.0 up
[root@localhost ~]# route add -host 192.168.78.5 dev eth0:2
```

在上述命令中，ifconfig 的参数 eth0:1 用于为网络配置文件 eth0 设置别名，eth0 后为

任意正整数，取值范围为 0～255；“broadcast<IP 地址>”用于为指定网卡设置广播协议，“netmask<子网掩码>”用于设置网卡的子网掩码，up 用于启动指定的网卡，如 eth0:1。

route 命令的参数 add，用于添加增加路由的相关参数，-host 表示其参数值连接到单个主机的路由地址，dev 用于指定该路由通过哪一块网卡连线出去，如 eth0:1。

配置完成后即可通过图 4-24 或图 4-25 的方式进行验证。值得一提的是，通过 ifconfig 和 route 配置的 IP 别名在系统网络服务重启后就会消失。

多学一招：开机自启动 ifconfig 和 route 命令

由于在 Linux 中使用 ifconfig 和 route 命令执行的相关操作，在 reboot 系统后，就会自动消失。为了解决这个问题可以将 ifconfig 和 route 命令添加到/etc/rc.local 文件中，使系统开机时就会自动运行相关操作。

例如，在 IP 地址为 192.168.78.3 的网卡上设置别名。具体操作步骤如下。

(1) 编写/etc/rc.local。

```
[root@localhost ~]#vi /etc/rc.local
```

打开文件后，在文件末尾添加以下配置指令，保存即可。

```
ifconfig eth0:1 192.168.78.4 broadcast 192.168.78.255 netmask 255.255.255.0 up
route add -host 192.168.78.4 dev eth0:1
```

(2) 验证测试。

为了验证 ifconfig 和 route 命令设置的开机自启动功能，重新启动 Linux 系统后，即可通过图 4-24 或图 4-25 的方式进行测试。

2. 配置基于 IP 的虚拟主机

接下来，基于 IP 为 192.168.78.3 的虚拟机以及其别名为 192.168.78.4 的 IP 配置 Nginx 的虚拟主机。

(1) 修改 nginx.conf 配置文件，在 http 块中添加以下两个 server 配置，具体如下。

```
#配置基于 IP 为 192.168.78.3 的虚拟主机
server {
    listen              80;
    server_name         192.168.78.3;
    root                html/192.168.78.3;
    index               index.html index.htm;
 }
#配置基于 IP 为 192.168.78.4 的虚拟主机
server {
    listen              80;
    server_name         192.168.78.4;
    root                html/192.168.78.4;
    index               index.html index.htm;
}
```

在上述配置中，为了便于测试，将 IP 地址为 192.168.78.3 和 192.168.78.4 的网站根

目录设置在 html 目录下与 IP 地址同名的目录中。完成上述配置后，保存 nginx.conf，平滑重启 Nginx 使设置生效。

(2) 编写测试文件并查看结果。

首先，在/usr/local/nginx/html/下分别创建目录 192.168.78.3 和 192.168.78.4。然后，在不同网站根目录下放置一个测试文件用于访问测试。

① 在 192.168.78.3 目录中创建 index.html 测试文件，编写内容如下。

```
<h1>Welcome 192.168.78.3!</h1>
```

② 在 192.168.78.4 目录中创建 index.html 测试文件，编写内容如下。

```
<h1>Welcome 192.168.78.4!</h1>
```

最后，在浏览器中分别访问 http://192.168.78.3 和 http://192.168.78.4 进行测试。成功后效果如图 4-26 和图 4-27 所示。

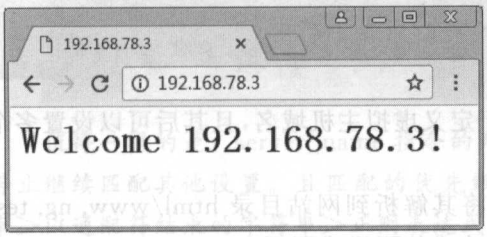


图 4-26　IP 为 192.168.78.3 的虚拟主机访问测试

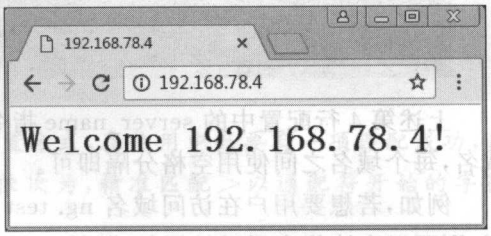


图 4-27　IP 为 192.168.78.4 的虚拟主机访问测试

4.4.4　基于域名配置虚拟主机

在真实的上线环境中，一个网站需要域名和公网 IP 地址才可以访问，但是申请域名和 IP 比较麻烦，且需要支付一定的费用。为了便于学习和测试，可以利用系统提供的 hosts 文件来设置一个虚拟的域名，并将域名解析到指定 IP 地址。

(1) 修改 hosts 文件，实现网站的域名访问。

首先，需要以“管理员身份运行”文本编辑器，打开在 C:\Windows\System32\drivers\etc 目录下的 hosts 文件。若没有管理员权限，则文件修改后将无法保存。

接着，在 hosts 文件中完成 IP 地址与域名映射的配置，具体如下。

```
192.168.78.3 www.ng.test
192.168.78.3 ng.test
```

根据域名的使用习惯，域名 www.ng.test 和 ng.test 在访问时通常指向同一网站。例如，域名 www.itheima.com 与 itheima.com 指的都是黑马程序员网站。

按照上述方式配置后，用户通过 Windows 中的浏览器，访问域名为 www.ng.test 或 ng.test 的网站时，就会自动解析到 IP 地址为 192.168.78.3 的服务器上。

值得一提的是，若用户在 Linux 系统下访问该域名，则需要编辑/etc/目录的 hosts 文件，具体如下。

```
[root@localhost ~]#vi /etc/hosts
```

在打开的 hosts 文件中添加以上 IP 地址与域名映射的配置即可。若在当前虚拟机中访问该域名，IP 地址可以使用 127.0.0.1。

```
127.0.0.1  www.ng.test
127.0.0.1  ng.test
```

因此，具体配置哪个系统下的 hosts 文件，要看在哪访问配置的域名。这里以 Windows 中的 hosts 为例，在其下的浏览器中访问测试。

(2) 打开 nginx.conf 配置文件，在 http 块中添加以下配置，实现基于域名的虚拟主机。

```
1   #配置域名为 www.ng.test 的虚拟主机
2   server {
3         listen              80;
4         server_name         www.ng.test;
5         root                html/www.ng.test;
6         index               index.html index.htm;
7   }
```

上述第 4 行配置中的 server_name 指令，用于定义虚拟主机域名，且其后可以设置多个域名，每个域名之间使用空格分隔即可。

例如，若想要用户在访问域名 ng.test 时，也将其解析到网站目录 html/www.ng.test 下，则可以在该指令后继续添加，将第 4 行配置修改成如下形式即可。

```
server_name   ng.test  www.ng.test;
```

完成上述配置后，保存 nginx.conf，平滑重启 Nginx 使设置生效。

(3) 编写测试文件并查看结果。

在/usr/local/nginx/html/下创建目录 www.ng.test。然后，在该网站根目录下放置一个测试文件 index.html，内容如下。

```
<h1>Welcome www.ng.test!</h1>
```

在浏览器中分别访问 http://www.ng.test 和 http://ng.test 进行测试。成功后效果如图 4-28 和图 4-29 所示。从图中可以看到，两个域名都解析到同一网站根目录下，页面的提示信息相同。

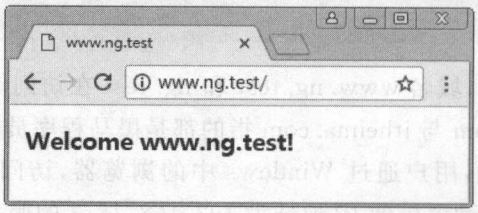


图 4-28 域名为 www.ng.test 的虚拟主机测试

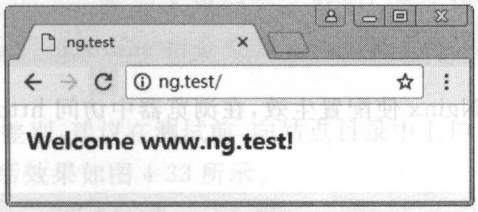


图 4-29　域名为 ng. test 的虚拟主机测试

多学一招：server_name 指令的使用

server_name 指令除了上述讲解到的精准配置方式外，还可利用通配符（*）与正则表达式设置域名，实现域名的泛解析。具体使用示例如下。

```
#以 * 通配符开始的字符串
server_name  * .test.com;
#以 * 通配符结束的字符串
server_name  www.* ;
#匹配正则表达式
server_name  ~^(?.+)\.domain\.com$;
```

值得一提的是，server_name 指令的几种设置方式，在使用中只要有一项匹配成功，则停止继续匹配其他设置。且匹配的优先级顺序依次为，精准匹配＞以通配符开始的字符串＞以通配符结束的字符串＞正则表达式。

4.4.5　设置目录列表

Nginx 默认是不允许列出整个目录的，所以，当用户访问某一站点或目录，且该站点或目录下没有 index 指令设置的默认索引文件（如 index. html 等）时，就会报 403 Forbidden 错误。但是当开启了目录列表功能后，出现上述的情况就可以让该站点或目录下的文件以列表的形式展示。

在 Nginx 中开启目录列表功能非常简单，只需要配置 autoindex 指令即可。具体使用方式如下。

```
autoindex on;
```

上述指令在不同块中的作用范围也不同，在 http 块中，表示用于对所有站点都有效；在 server 块中，表示对指定站点有效；在 location 块中，表示对某个目录起作用。

为了让读者更加清晰地了解 autoindex 指令的使用，下面基于 IP 为 192. 168. 78. 3 的虚拟主机进行演示。

（1）配置虚拟主机，且该虚拟主机的站点根目录下没有指定的默认索引文件，具体如下。

```
server {
    listen 80;
    server_name 192.168.78.3;
    root  html;
```

```
    index index.php;
}
```

设置完成后，可重启 Nginx 使配置生效，在浏览器中访问 http://192.168.78.3 的效果如图 4-30 所示。

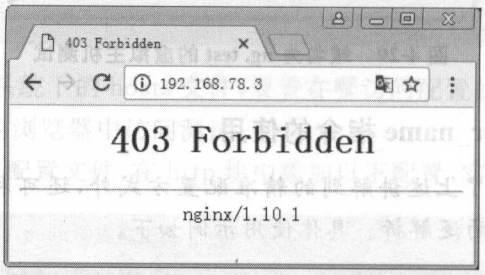


图 4-30 默认目录列表设置

(2) 设置目录列表。

在上一步的配置中添加以下指令，完成目录列表的设置。具体如下。

```
autoindex on;
```

设置完成，平滑重启 Nginx 使配置生效。再次访问 http://192.168.78.3，如图 4-31 所示。从图中可以看出，该站点目录下没有指定的默认索引文件 index.php，从而显示该站点下的目录列表。除了目录中的文件名称外，还包括各文件最后一次修改的时间(格林尼治时间 GMT)和文件的大小(默认是以字节 bytes 为单位的准确值)。

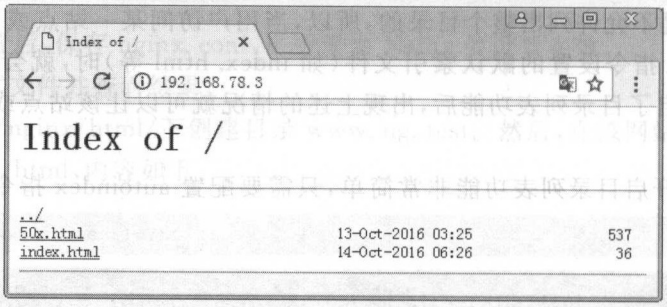


图 4-31 目录列表

(3) 设置显示文件的时间格式与大小。

在开启目录列表功能时，还可以通过 Nginx 提供的 autoindex_exact_size 指令设置精准显示文件大小还是大概显示文件大小；通过 autoindex_localtime 指令设置文件最后一次修改时间的格式。默认情况下，autoindex_exact_size 指令和 autoindex_localtime 指令的值分别为 on 和 off，效果如图 4-31 所示。

接下来，在步骤(2)的配置文件中添加以下两条指令设置。

```
autoindex_exact_size off;
autoindex_localtime on;
```

在上述配置中，autoindex_exact_size 指令设置为 off，表示以 kB/MB/GB 为单位显示文件的大概大小；autoindex_localtime 指令设置为 on，表示显示的时间为文件的服务器时间。

为了更加明显地看到差别，建议在测试前，向站点目录中上传大文件进行对比。设置前效果如图 4-32 所示，设置后效果如图 4-33 所示。

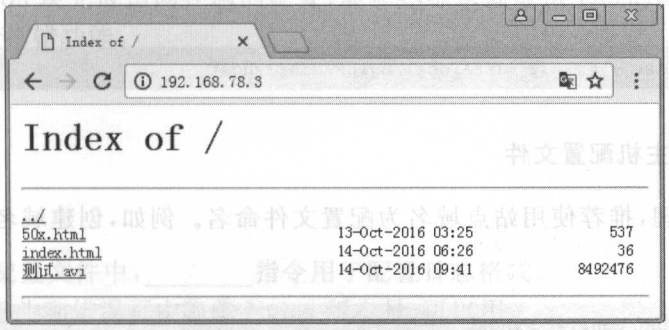


图 4-32　设置前文件时间与大小

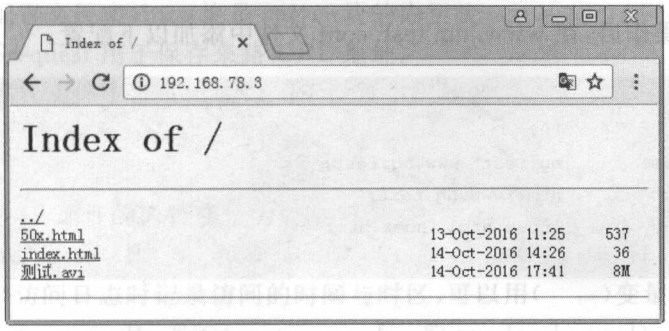


图 4-33　设置后文件时间与大小

对比后可以看出，添加上述两条指令后，文件的显示时间与文件大小的显示形式更易于阅读理解。因此推荐设置目录列表的同时添加指令 autoindex_exact_size 和 autoindex_localtime。

4.4.6　子配置文件的引入

由于一个 Nginx 服务器可运行多个虚拟主机，但如果将所有虚拟主机的配置全部放在 nginx.conf 文件中，则会造成 nginx.conf 文件过大、可读性差，对日后的维护带来诸多不便。因此，Nginx 中提供了 include 指令用于组织和管理 Nginx 相关的配置信息。

include 指令的具体语法如下所示。

```
include file | mask;
```

在上述语法中，*file* 用于指定包含的文件名称，mask 用于指定某一路径下的文件，其路径可以是相对路径，也可以是绝对路径。其中，在使用相对路径的情况下，相对的路径是 Nginx 的安装路径下的 conf 目录/usr/local/nginx/conf。

接下来，为了使读者更好地了解include指令的使用，下面对上述基于域名配置的虚拟主机进行修改。具体步骤如下。

1. 创建目录

在/usr/local/nginx/conf路径下创建vhost目录，用于保存Nginx服务器的虚拟主机配置文件。

```
[root@localhost ~]#mkdir /usr/local/nginx/conf/vhost
```

2. 编写虚拟主机配置文件

为了便于管理，推荐使用站点域名为配置文件命名。例如，创建域名为www.ng.test的配置文件。

```
[root@localhost conf]#cd vhost
[root@localhost vhost]#touch www.ng.test.conf
[root@localhost vhost]#vi www.ng.test.conf
```

执行完上述操作后，在www.ng.test.conf文件中添加以下配置。

```
server {
    listen          80;
    server_name     ng.test  www.ng.test;
    root            html/www.ng.test;
    index           index.html index.htm;
}
```

3. include引入配置文件

打开Nginx的主配置文件nginx.conf，删除相关的虚拟主机配置，在http块中利用include指令完成www.ng.test.conf文件的引入。具体配置如下。

```
#第1种方式:单个文件引入
include vhost/www.ng.test.conf;
#第2种方式:利用通配符
include vhost/*.conf;
```

上述配置中，给出了两种引入文件的方式，当引入的文件数目少时，推荐使用方式1，按照精准的文件名称引入；当引用文件数目过多时，为了方便维护管理，推荐使用方式2，利用通配符引入vhost目录下所有的配置文件。

4. 验证测试

按照上述步骤操作完成后，平滑重启Nginx使配置生效。访问http://www.ng.test和http://ng.test进行测试，若查看的结果与图4-28和图4-29相同，则表明include指令引入子配置文件成功。

本章小结

本章主要介绍 Nginx 的基本配置，要求读者能够了解配置文件的结构、默认配置文件中指令的含义，工作进程的用户和组配置，以及如何自定义错误页面；能够熟练使用 allow、deny 指令与 location 块完成访问控制的配置；能够熟练掌握虚拟主机的配置，灵活地运用日志文件实现统计排错功能。

课后练习

一、填空题

1. 在 Nginx 配置文件中，_____指令用于配置日志格式。
2. 在响应消息头和错误页中隐藏 Nginx 版本号，可以用_____指令。

二、判断题

1. 在 Nginx 配置文件中，http 块是 server 块的内层块。（　　）
2. 内置变量 $request 用于保存来路 URL 地址。（　　）
3. 在 Nginx 配置文件中，listen 指令用于设置主机域名。（　　）

三、选择题

1. 获取客户端 IP 地址的内置变量为（　　）。
   A. $remote_user　B. $remote_addr　C. $request　D. $http_referer
2. 若要在记录访问日志时记录访问的时间与时区，可以用（　　）变量。
   A. $request　B. $status　C. $time_local　D. $http_referer

四、简答题

1. 简述 location / {} 与 location =/ {} 的区别。
2. 简述 location 的前缀～、～ *、^～的区别。

五、操作题

SSI(Server Side Include)称为“服务器端包含”或“服务器端嵌入”技术，是一种基于服务器端的网页制作技术，它可以将文本、图形或应用程序信息包含到网页中。Nginx 服务器中也添加了对 SSI 应用的支持。下面请在 Nginx 中开启 SSI，通过编写 shtml 文件实现显示当前请求的时间和 IP。

第5章 Web服务器搭建

学习目标

- 掌握 Nginx+PHP 环境的搭建和配置；
- 掌握 Nginx 与 Apache、Tomcat 实现动静分离；
- 掌握 OpenResty 环境的搭建与使用。

Nginx 的主要用途是作为 Web 服务器使用，目前最典型的应用是与 PHP、Tomcat、MySQL 等软件组成动态网站平台。本章将讲解 Nginx 与各种软件组成的 Web 服务器环境如何搭配，包括各种软件的安装与配置，以及基于 Nginx+Lua 的高性能 Web 平台 OpenResty 的安装与使用。

5.1 Nginx+PHP 环境

5.1.1 PHP 的安装与使用

PHP(Hypertext Preprocessor，超文本预处理器)是一种运行于服务器端的嵌入式脚本编程语言，具有开源免费、易学易用、开发效率高等特点，是 Web 应用开发的主流语言之一。目前由 LAMP(Linux、Apache、PHP、MySQL)组成的平台现已被大量应用在网站系统的搭建中。

在 LAMP 环境中，Apache 作为 Web 服务器与客户端浏览器交互，PHP 负责处理复杂的网站业务逻辑需求，MySQL 负责存储和管理网站的数据库。而后起之秀的 Nginx，在 Web 服务器功能方面可以取代 Apache 组成 LNMP 平台。关于整体的 LNMP 平台会在后面的章节中详细讲解，本节重点介绍 Nginx 与 PHP 的整合，从而形成一个基本的动态网站运行环境。

1. 获取 PHP

在 PHP 的官方网站 http://php.net 可以获取 PHP 源代码的下载地址，如图 5-1 所示。目前 PHP 官方网站发布了 5.6 和 7.0 两种版本，其中 5.6 具有很强的兼容性，而 7.0 具有优越的性能。

本书选择以 PHP 5.6.27 版本进行讲解，在网站中找到 php-5.6.27.tar.gz 压缩包的下载地址，下载到 Linux 服务器中。使用 tar -zxvf 命令进行解压，然后查看解压后的文件列表，如图 5-2 所示。

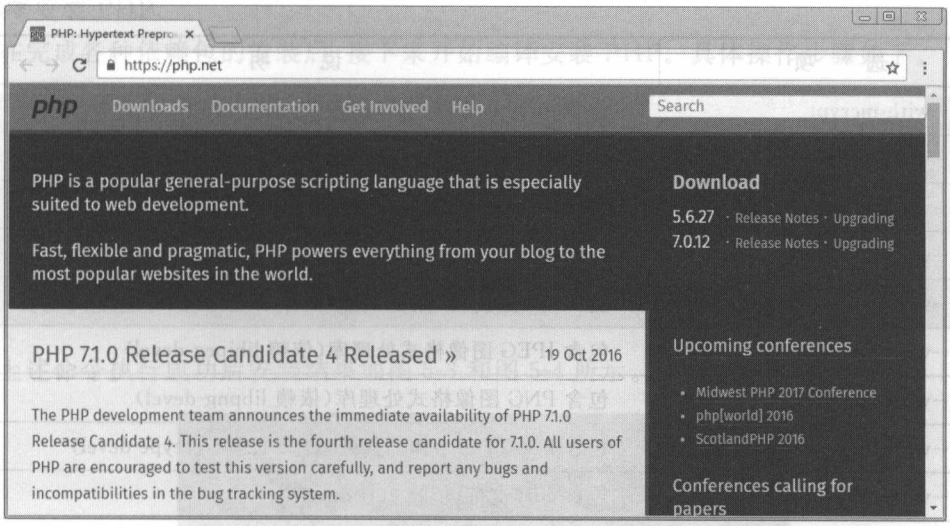


图 5-1　PHP 网站首页

```
[root@localhost php-5.6.27]# ls
acinclude.m4      generated_lists  mkinstalldirs          README.NEW-OUTPUT-API          server-tests-config.php
aclocal.m4        genfiles         netware                README.PARAMETER_PARSING_API   server-tests.php
build             header           NEWS                   README.REDIST.BINS             snapshot
buildconf         INSTALL          pear                   README.RELEASE_PROCESS         stamp-h.in
buildconf.bat     install-sh       php5.spec.in           README.SELF-CONTAINED-EXTENSIONS stub.c
CODING_STANDARDS  LICENSE          php.gif                README.STREAMS                 tests
config.guess      ltmain.sh        php.ini-development    README.SUBMITTING_PATCH        travis
config.sub        main             php.ini-production     README.TESTING                 TSRM
configure         makedist         README.EXT_SKEL        README.TESTING2                UPGRADING
configure.in      Makefile.frag    README.GIT-RULES       README.UNIX-BUILD-SYSTEM       UPGRADING.INTERNALS
CREDITS           Makefile.gcov    README.input_filter    README.WIN32-BUILD-SYSTEM      vcsclean
ext               Makefile.global  README.MAILINGLIST_RULES run-tests.php              win32
EXTENSIONS        makerpm          README.md              sapi                           Zend
footer            missing          README.namespaces      scripts
[root@localhost php-5.6.27]#
```

图 5-2　PHP 解压后的文件列表

2. 编译安装 PHP

从解压后的目录中可以看出，PHP 提供 configure 程序用于编译安装。使用 ./configure --help 命令可以查看详细的编译选项，也可以查看 PHP 官方手册。对于一般用户来说，无须彻底明白这些编译选项的作用，只需了解几个常用选项即可安装使用，具体如表 5-1 所示。

表 5-1　PHP5.6 常用编译选项

选　项	说　明
--prefix	安装目录，默认目录为/usr/local，也可以设为/usr/local/php
--enable-fpm	开启 PHP 的 FPM 功能，提供 PHP FastCGI 管理器
--with-zlib	包含 zlib 库，支持数据压缩和解压缩
--enable-zip	开启 ZIP 功能
--enable-mbstring	开启 mbstring 功能，用于多字节字符串处理

续表

选　项	说　明
--with-mcrypt	包含 mcrypt 加密支持(依赖 libmcrypt)
--with-mysql	包含 MySQL 数据库访问支持
--with-mysqli	包含增强版的 MySQL 数据库访问支持
--with-pdo-mysql	包含基于 PDO(PHP Data Object)的 MySQL 数据库访问支持
--with-gd	包含 GD 库支持,用于 PHP 图像处理
--with-jpeg-dir	包含 JPEG 图像格式处理库(依赖 libjpeg-devel)
--with-png-dir	包含 PNG 图像格式处理库(依赖 libpng-devel)
--with-freetype-dir	包含 FreeType 字体图像处理库(依赖 freetype-devel)
--with-curl	包含 curl 支持(依赖 curl-devel)
--with-openssl	包含 OpenSSL 支持(依赖 openssl-devel)
--with-mhash	包含 mhash 加密支持
--enable-bcmath	开启精准计算功能
--enable-opcache	开启 opcache 功能,一种 PHP 的代码优化器

在上述编译选项中,有些选项的前缀是 enable,有些是 with,其区别在于 enable 选项用于开启 PHP 的一些内置的功能,而 with 选项依赖于系统中的共享库,如果系统中没有则需要安装依赖包。

表 5-1 中列举的是运行 PHP 大部分成熟项目所需要的扩展。在实际使用时,用户也可以根据需要自行定制。推荐读者在学习阶段选择典型的扩展,并安装所需的依赖包。下面介绍详细的安装步骤。

1) 通过 yum 安装依赖

PHP 的大部分依赖可以通过 yum 自动安装。具体命令如下。

```
[root@localhost ~]#yum -y install libxml2-devel openssl-devel \
curl-devel libjpeg-devel libpng-devel freetype-devel
```

其中,libxml2-devel 是 PHP 编译安装所必需的依赖包,其余的是 PHP 各种扩展的依赖包。

2) 安装 libmcrypt 依赖

目前 yum 中没有 libmcrypt,需要手动下载安装。libmcrypt 的源代码可以在开源软件平台 SourceForge 网站中获取,项目地址是 https://sourceforge.net/projects/mcrypt,下载后按照如下命令安装即可。

```
[root@localhost ~]#tar -zxvf libmcrypt-2.5.8.tar.gz
[root@localhost ~]#cd libmcrypt-2.5.8
[root@localhost libmcrypt-2.5.8]#./configure
[root@localhost libmcrypt-2.5.8]#make && make install
```

3）安装 PHP

在完成各种依赖包的安装后，接下来开始编译安装 PHP。具体操作步骤如下。

```
[root@localhost ~]#cd php-5.6.27
[root@localhost php-5.6.27]#./configure --prefix=/usr/local/php --enable-fpm \
--with-zlib --enable-zip --enable-mbstring --with-mcrypt --with-mysql \
--with-mysqli --with-pdo-mysql --with-gd --with-jpeg-dir --with-png-dir \
--with-freetype-dir --with-curl --with-openssl --with-mhash --enable-bcmath \
--enable-opcache
[root@localhost php-5.6.27]#make && make install
```

上述命令执行成功后安装结果如图 5-3 和图 5-4 所示。

```
+--------------------------------------------------------------------+
| License:                                                           |
| This software is subject to the PHP License, available in this     |
| distribution in the file LICENSE.  By continuing this installation |
| process, you are bound by the terms of this license agreement.     |
| If you do not agree with the terms of this license, you must abort |
| the installation process at this point.                            |
+--------------------------------------------------------------------+

Thank you for using PHP.

config.status: creating php5.spec
config.status: creating main/build-defs.h
config.status: creating scripts/phpize
config.status: creating scripts/man1/phpize.1
config.status: creating scripts/php-config
config.status: creating scripts/man1/php-config.1
config.status: creating sapi/cli/php.1
config.status: creating sapi/fpm/php-fpm.conf
config.status: creating sapi/fpm/init.d.php-fpm
config.status: creating sapi/fpm/php-fpm.service
config.status: creating sapi/fpm/php-fpm.8
config.status: creating sapi/fpm/status.html
config.status: creating sapi/cgi/php-cgi.1
config.status: creating ext/phar/phar.1
config.status: creating ext/phar/phar.phar.1
config.status: creating main/php_config.h
config.status: executing default commands
[root@localhost php-5.6.27]#
```

图 5-3　PHP 编译前的配置

```
Installing PHP CLI binary:        /usr/local/php/bin/
Installing PHP CLI man page:      /usr/local/php/php/man/man1/
Installing PHP FPM binary:        /usr/local/php/sbin/
Installing PHP FPM config:        /usr/local/php/etc/
Installing PHP FPM man page:      /usr/local/php/php/man/man8/
Installing PHP FPM status page:   /usr/local/php/php/php/fpm/
Installing PHP CGI binary:        /usr/local/php/bin/
Installing PHP CGI man page:      /usr/local/php/php/man/man1/
Installing build environment:     /usr/local/php/lib/php/build/
Installing header files:          /usr/local/php/include/php/
Installing helper programs:       /usr/local/php/bin/
  program: phpize
  program: php-config
Installing man pages:             /usr/local/php/php/man/man1/
  page: phpize.1
  page: php-config.1
Installing PEAR environment:      /usr/local/php/lib/php/
[PEAR] Archive_Tar    - installed: 1.4.0
[PEAR] Console_Getopt - installed: 1.4.1
[PEAR] Structures_Graph- installed: 1.1.1
[PEAR] XML_Util       - installed: 1.3.0
[PEAR] PEAR           - installed: 1.10.1
Wrote PEAR system config file at: /usr/local/php/etc/pear.conf
You may want to add: /usr/local/php/lib/php to your php.ini include_path
/root/php-5.6.27/build/shtool install -c ext/phar/phar.phar /usr/local/php/bin
ln -s -f phar.phar /usr/local/php/bin/phar
Installing PDO headers:           /usr/local/php/include/php/ext/pdo/
[root@localhost php-5.6.27]#
```

图 5-4　PHP 编译安装

3. PHP的简单使用

在Linux系统中安装PHP后即可通过命令的方式来使用，用户也可以将PHP代码写在.php脚本文件中，由PHP程序执行。下面分别介绍两种方式的使用方法。

1）通过.php文件执行PHP代码

编写.php文件是在PHP网站开发领域使用最多的方法，通常一个完整的PHP网站项目会由成百上千个.php文件组成，其代码遵循PHP语言的语法。为了测试安装后的PHP是否能够正常运行，接下来编写一个基本的Hello World程序。执行vi ~/test.php命令新建文件，编写PHP代码如下。

```
<?php
    echo "Hello World !\n";
?>
```

在上述代码中，开始和结束的<?php ?>是PHP的标记。PHP是嵌入到HTML中的语言，需要将PHP代码写在标记内，而标记外面可以写HTML内容。第2行代码中的echo语句用于输出内容，后面是一个双引号字符串，最后以分号结束。字符串中的\n是一个用转义字符方式表示的换行符。

将文件保存后，利用PHP安装目录下的可执行程序来执行.php文件，具体命令如下。

```
[root@localhost ~]#/usr/local/php/bin/php -f test.php
Hello World !
```

从上述结果可以看到，PHP成功执行了test.php文件，输出Hello World！运行结果。其中选项-f表示执行给定的PHP脚本文件。

2）直接执行PHP代码

通过./php -r的方式可以将PHP代码直接交给PHP程序执行，具体示例如下。

```
[root@localhost ~]#cd /usr/local/php/bin
[root@localhost bin]#./php -r 'echo 100+200,"\n";'
300
```

从上述执行结果可以看出，PHP输出给定代码中的表达式100+200的计算结果300。

关于PHP编程语言的详细语法，这里不再展开讲解，读者可以参考其他资料进行学习。PHP是Web开发的常用语言，使用非常方便，本书后面的一些示例中也会用到PHP语言进行演示。

5.1.2 PHP与Nginx整合

1. 认识FastCGI

实现PHP与Nginx的整合，离不开FastCGI。对于Nginx而言，PHP是一个外部程序，而非Nginx内部的一个模块。由于Linux平台下可以有多种不同的Web服务器和应用程序，为了让Web服务器的功能扩展性更强，就需要支持CGI（Common Gateway

Interface，公共网关接口）规范。

CGI 是 Web 服务器与外部程序（即 CGI 程序）之间的接口标准，用于两种不同程序之间的信息传递。CGI 规范允许 Web 服务器根据浏览器请求调用 CGI 程序，并将其输出结果通过响应发送给浏览器，从而使 Web 服务器支持处理复杂的网站业务需求。

Web 服务器支持 CGI 的意义不在于性能而在于开发速度。例如，一个功能复杂且需求多变的网站，如果用 Nginx 模块（基于 C 语言）的方式来开发成本将非常高，而 PHP 这样的 CGI 程序就是为快速开发而设计的，虽然有性能损失，但其带来的好处远远大于这些损失。图 5-5 演示了 Web 服务器和 CGI 程序的工作流程，开发者只需要编写一个 CGI 文件放在网站目录中，当浏览器请求 CGI 文件时，Web 服务器就会调用 CGI 程序执行 CGI 文件，等待 CGI 程序处理完成后，再将程序的输出结果返回给浏览器。

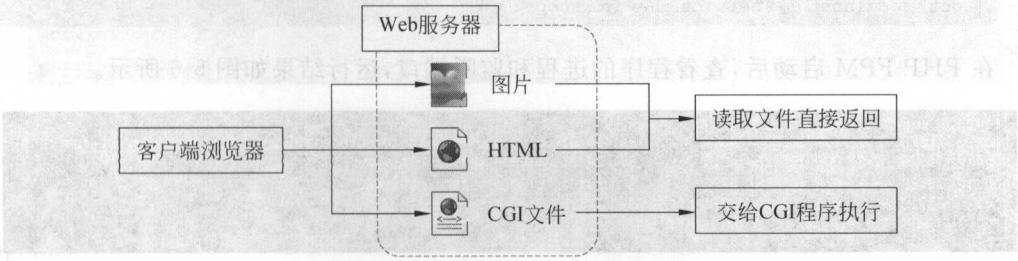


图 5-5　请求处理流程

在实际配置 Nginx 时，可以利用 Location 规则，实现根据不同请求 URI 采取不同的处理方式。例如，当用户请求一个 .php 脚本文件时，就调用 PHP 程序来运行该脚本，然后把 PHP 执行后的输出结果发送给浏览器。

Nginx 和 PHP 还支持 CGI 的改良版——FastCGI，主要用于解决 CGI 性能上的缺陷。传统 CGI 方式是每当客户端请求 CGI 时，Web 服务器就通过操作系统创建一个新的 CGI 进程，一个 CGI 进程完成一个请求处理后就退出，下次请求再创建一个新 CGI 进程。由于这种方式需要不断为每个请求创建进程，因此在网站并发量大时显得非常低效。FastCGI 优化了这种工作方式，它由一个常驻的 CGI 进程管理器，通过管理一个进程池来处理 Web 服务器的请求，由此提高了性能。

2. PHP-FPM

PHP 提供的 PHP-FPM（FastCGI Process Manager）就是一个 FastCGI 进程管理器，其可执行文件位于 PHP 安装目录下的 sbin 目录中。在启动 PHP-FPM 之前，需要先创建配置文件。PHP 在 etc 目录下提供了默认配置文件 php-fpm. conf. default，将该文件复制为 php-fpm. conf 即可。具体操作步骤如下。

```
[root@localhost ~]#cd /usr/local/php/etc
[root@localhost etc]#ls
pear.conf  php-fpm.conf.default
[root@localhost etc]#cp php-fpm.conf.default php-fpm.conf
```

虽然执行 sbin 目录下的 php-fpm 可执行文件可以启动 PHP-FPM，但这种方式比较麻

烦。PHP 在源码包中提供了 service 方式管理 PHP-FPM 的 shell 脚本，下面将脚本文件复制到/etc/init.d 目录中，并通过 chkconfig 实现开机启动，具体步骤如下。

```
[root@localhost etc]#cd ~/php-5.6.27
[root@localhost php-5.6.27]#cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
[root@localhost php-5.6.27]#chmod +x /etc/init.d/php-fpm
[root@localhost php-5.6.27]#chkconfig --add php-fpm
```

接下来就可以通过 service 方式管理 PHP-FPM，实现启动、重启或停止服务。

```
[root@localhost ~]#service php-fpm start
[root@localhost ~]#service php-fpm reload
[root@localhost ~]#service php-fpm restart
[root@localhost ~]#service php-fpm stop
```

在 PHP-FPM 启动后，查看程序的进程和监听端口，运行结果如图 5-6 所示。

```
[root@localhost ~]# ps aux | grep php
root     15878  0.0  0.9 105868  4804 ?      Ss   14:31   0:00 php-fpm: master process (/usr/local/php/etc/php-fpm.conf)
nobody   15879  0.0  0.8 105868  4216 ?      S    14:31   0:00 php-fpm: pool www
nobody   15880  0.0  0.8 105868  4216 ?      S    14:31   0:00 php-fpm: pool www
root     15910  0.0  0.1 103312   876 pts/0  S+   14:35   0:00 grep php
[root@localhost ~]# netstat -tlnp | grep php
tcp        0      0 127.0.0.1:9000              0.0.0.0:*                   LISTEN      15878/php-fpm
```

图 5-6 查看 PHP-FPM 进程和监听端口

从图中可以看出，PHP-FPM 的进程已经启动，并且监听了 9000 端口。PHP-FPM 的主进程用户是 root，子进程工作于 nobody 用户。

3. PHP 配置文件

PHP 的配置文件主要包括 php-fpm.conf 和 php.ini，下面分别进行讲解。需要注意的是，在更改配置文件后，需要重启 PHP-FPM 服务，才可以使配置生效。

1）php-fpm.conf

PHP-FPM 启动时，具体监听的端口号和工作用户在 php-fpm.conf 配置文件中可以修改。该配置文件主要为熟悉 PHP-FPM 工作方式的用户根据服务器的性能进行优化。下面列举一些简单的配置说明。

```
[global]                                [全局配置]
;pid=run/php-fpm.pid                    保存 pid 到文件，默认不保存
;error_log=log/php-fpm.log              指定错误日志保存目录，默认在 PHP 的 var 目录
;daemonize=yes                          是否作为后台守护进程启动 FPM，默认为 yes
;events.mechanism=epoll                 使用的事件机制，默认自动检测
[www]                                   [www 进程池配置]
user=nobody                             工作用户
group=nobody                            工作组
listen=127.0.0.1:9000                   监听的端口，也可以是 socket 文件（如/tmp/php-cgi.sock）
;listen.owner=nobody                    socket 文件的所属用户
;listen.group=nobody                    socket 文件的所属组
;listen.mode=0660                       socket 文件的权限（以 0 开始的八进制数）
```

```
;listen.allowed_clients=127.0.0.1          指定允许连接的客户端 IP,默认为任意的
pm=dynamic                                 控制子进程的数量,默认为 dynamic(动态控制)
;access.log=log/$pool.access.log           访问日志文件,默认为不记录日志
;php_flag[display_errors]=off              以 php_flag 方式覆盖 php.ini 中的配置
;php_admin_value[memory_limit]=32M         以 php_admin_value 方式覆盖 php.ini 中的配置
```

在上述配置中,以分号“;”开始的配置是注释,表示该配置没有生效,将自动使用默认值。

2) php.ini

前面介绍的 php-fpm.conf 只和 PHP-FPM 有关,而 PHP 本身也有一个配置文件 php.ini。默认情况下,PHP 会到/usr/local/php/lib 目录中搜索 php.ini 文件,但是该文件默认并没有被安装。在 PHP 解压后的源码包中可以找到两个预设的 php.ini 文件,如下所示。

```
[root@localhost php-5.6.27]#ls | grep php.ini
php.ini-development
php.ini-production
```

这两个预设文件分别用于不同场合,php.ini-development 适合开发环境(开发项目时方便测试程序),php.ini-production 适合实际上线环境(安全性较高)。

任选一种预设配置复制到 PHP 的 lib 目录中即可使用,这里推荐选择开发环境的 php.ini。

```
[root@localhost php-5.6.27]#cp php.ini-development /usr/local/php/lib/php.ini
```

在 php.ini 文件中也有许多复杂的配置,主要包括 PHP 的核心配置及各种扩展模块的配置。下面列举一些常用的配置说明。

```
[PHP]                                      [PHP 核心配置]
output_buffering=4096                      输出缓冲(字节数)
;open_basedir =                            限制 PHP 脚本可访问文件的路径
disable_functions =                        禁用的函数列表
max_execution_time=30                      每个 PHP 脚本最长时间限制(秒数)
memory_limit=128M                          每个 PHP 脚本最大内存使用限制
display_errors=On                          是否输出错误信息
log_errors=On                              是否开启错误日志
;error_log=php_errors.log                  错误日志保存位置
post_max_size=8M                           通过 POST 提交的最大限制
default_mimetype="text/html"               默认的 MIME 类型
default_charset="UTF-8"                    默认的字符集
file_uploads=On                            是否允许文件上传
;upload_tmp_dir =                          上传文件临时保存目录
upload_max_filesize=2M                     上传文件最大限制
allow_url_fopen=On                         是否允许打开远程文件
[Date]                                     [时间和日期配置]
;date.timezone =                           时区配置(如 UTC、PRC、Asia/Shanghai)
[mail function]                            [邮件配置]
;sendmail_path =                           sendmail 的路径
[Session]                                  [会话配置]
session.save_handler=files                 将会话以文件形式保存
;session.save_path="/tmp"                  回话保存目录
```

上述配置读者仅了解即可，在使用PHP的过程中，许多功能都和php.ini相关。尤其是在搭建LNMP环境时，不同项目对环境的要求不同，一旦配置不当会导致项目无法运行或出现安全问题。

4. FastCGI环境变量

正如浏览器与服务器之间通过HTTP协议交互时，双方会传递各自的环境信息，Nginx与PHP之间通过FastCGI交互时也需要传递一些信息，这些信息就是环境变量。在Nginx的conf目录中有一个fastcgi.conf文件，该文件中通过fastcgi_param数组型指令保存了一些环境变量，如下所示。

```
[root@localhost conf]#cat fastcgi.conf
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;
fastcgi_param  QUERY_STRING       $query_string;
fastcgi_param  REQUEST_METHOD     $request_method;
fastcgi_param  CONTENT_TYPE       $content_type;
fastcgi_param  CONTENT_LENGTH     $content_length;
⋮
```

在上述内容中，fastcgi_param指令的第1个参数是环境变量名称，第2个参数是对应的值。环境变量的“名称”和“值的格式”是由FastCGI接口规定的，主要包含客户端和Web服务器的环境信息，其常用环境变量如表5-2所示。

表5-2 FastCGI常用环境变量

选　项	说　明	示　例
SCRIPT_FILENAME	脚本文件路径	/usr/local/nginx/html/index.php
QUERY_STRING	？后面的URL参数	a=1&b=2
REQUEST_METHOD	请求方式(GET、POST)	POST
CONTENT_TYPE	请求内容的类型	application/x-www-form-urlencoded
CONTENT_LENGTH	请求内容的长度	8
SCRIPT_NAME	脚本文件名	/index.php
REQUEST_URI	请求URI	/index.php?a=1&b=1
DOCUMENT_URI	文档URI	/index.php
DOCUMENT_ROOT	文档根目录	/usr/local/nginx/html
SERVER_PROTOCOL	HTTP协议版本	HTTP/1.1
REQUEST_SCHEME	请求协议(http、https)	http
GATEWAY_INTERFACE	网关接口	CGI/1.1
SERVER_SOFTWARE	服务器软件和版本	nginx/1.10.1
REMOTE_ADDR	来源地址	192.168.78.1
REMOTE_PORT	来源端口	60100
SERVER_ADDR	服务器地址	192.168.78.3
SERVER_PORT	服务器端口	80
SERVER_NAME	服务器名称	ng.test

在 fastcgi. conf 配置文件中，只有 SCRIPT_FILENAME 是由 $document_root 和 $fastcgi_script_name 两个变量的值拼接成的。其余的皆与 Nginx 内置变量一一对应，例如 QUERY_STRING 对应 $query_string。

另外，在 Nginx 的 conf 目录还有一个 fastcgi_params 文件，该文件与 fastcgi. conf 唯一的区别是其缺少了 SCRIPT_FILENAME 的定义，由于历史原因 Nginx 保留了 fastcgi_params 这个文件。

5. 在 Nginx 配置文件中支持 PHP

在了解 FastCGI、PHP 的相关知识后，接下来分步骤讲解如何实现 Nginx 与 PHP 的交互。

(1) 以虚拟主机 www. ng. test 为例，修改配置文件如下。

```
1   server {
2       listen 80;
3       server_name ng.test www.ng.test;
4       root html/www.ng.test;
5       index index.html index.htm index.php;
6       location ~\.php$ {
7           fastcgi_pass 127.0.0.1:9000;
8           include fastcgi.conf;
9       }
10  }
```

在上述配置中，第 5 行 index 指令的参数后面增加了 index. php，用于支持 PHP 文件作为默认页面；第 6～9 行用于匹配路径以. php 结尾的请求，将这些请求发送给监听本机(127. 0. 0. 1)9000 端口的 FastCGI 程序(即 PHP)；第 8 行引入 FastCGI 的环境变量配置。

(2) 在 html/www. ng. test 目录中创建两个 PHP 文件用于测试，具体如下。

```
[root@localhost www.ng.test]#echo '<?php echo 123;' >test.php;
[root@localhost www.ng.test]#echo '<?php phpinfo();' >index.php;
[root@localhost www.ng.test]#cat test.php
<?php echo 123;
[root@localhost www.ng.test]#cat index.php
<?php phpinfo();
```

从上述操作可以看出，test. php 的文件内容是输出 123，index. php 的文件内容是调用 phpinfo()函数。其中，phpinfo()是 PHP 为显示其环境信息而提供的函数，通过它可以很方便地在网页中查看 PHP 的环境配置情况。

(3) 通过浏览器访问 http://www. ng. test/test. php，运行结果如图 5-7 所示。从图中可以看出，Web 服务器中的 PHP 已经正确执行了 test. php 脚本文件，并且由 Nginx 将执行结果 123 响应给浏览器。

(4) 通过浏览器访问 http://www. ng. test，运行结果如图 5-8 所示。该运行结果是 index. php 文件中的 phpinfo()函数的输出结果，这些内容包括系统信息(System)、编译日期(Build Date)、配置命令(Configure Command)、服务器 API(Server API)等基本信息，以

及各种模块的配置信息。

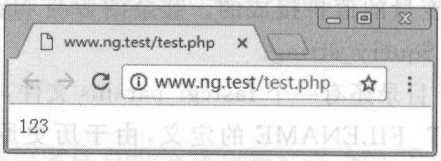
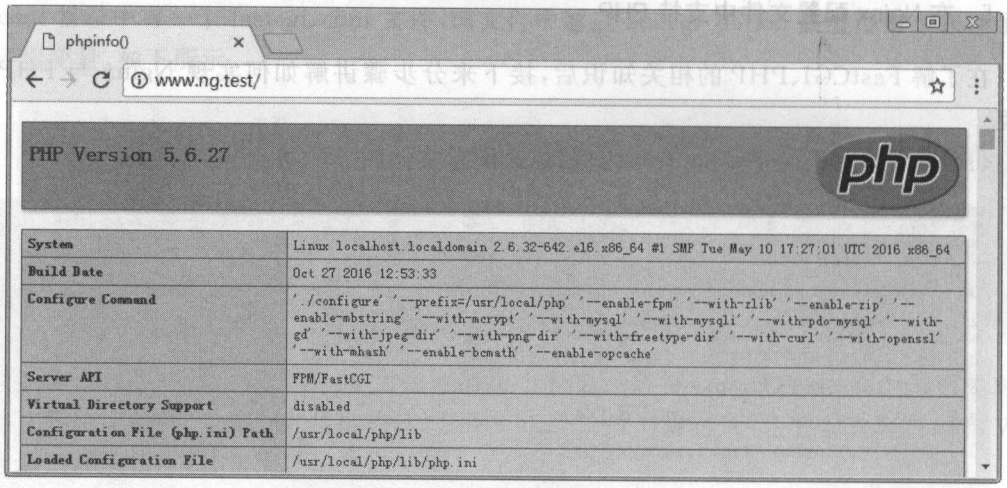


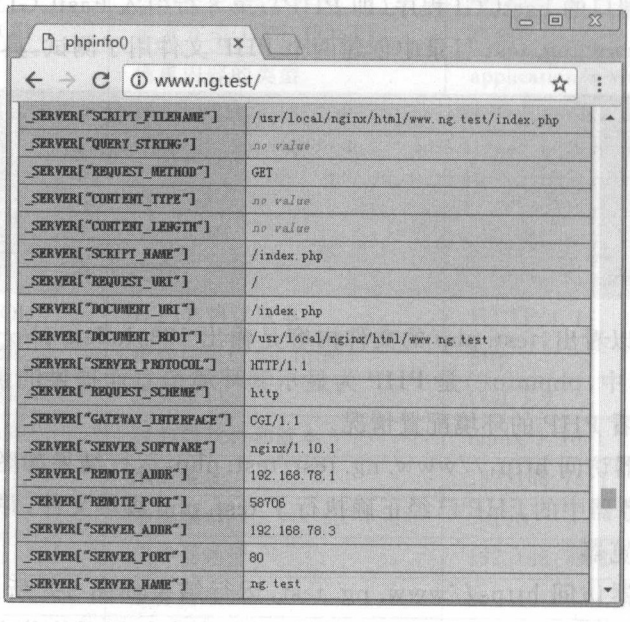
图 5-7 test.php 运行结果



System	Linux localhost.localdomain 2.6.32-642.el6.x86_64 #1 SMP Tue May 10 17:27:01 UTC 2016 x86_64
Build Date	Oct 27 2016 12:53:33
Configure Command	'./configure' '--prefix=/usr/local/php' '--enable-fpm' '--with-zlib' '--enable-zip' '--enable-mbstring' '--with-mcrypt' '--with-mysql' '--with-mysqli' '--with-pdo-mysql' '--with-gd' '--with-jpeg-dir' '--with-png-dir' '--with-freetype-dir' '--with-curl' '--with-openssl' '--with-mhash' '--enable-bcmath' '--enable-opcache'
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/php/lib
Loaded Configuration File	/usr/local/php/lib/php.ini

图 5-8 index.php 运行结果

值得一提的是，在浏览器中按 Ctrl＋F 键，搜索 SCRIPT_FILENAME，可以从 phpinfo 中找到 PHP 接收到的环境变量信息，如图 5-9 所示。这些信息在 PHP 脚本中都可以访问。



_SERVER["SCRIPT_FILENAME"]	/usr/local/nginx/html/www.ng.test/index.php
_SERVER["QUERY_STRING"]	*no value*
_SERVER["REQUEST_METHOD"]	GET
_SERVER["CONTENT_TYPE"]	*no value*
_SERVER["CONTENT_LENGTH"]	*no value*
_SERVER["SCRIPT_NAME"]	/index.php
_SERVER["REQUEST_URI"]	/
_SERVER["DOCUMENT_URI"]	/index.php
_SERVER["DOCUMENT_ROOT"]	/usr/local/nginx/html/www.ng.test
_SERVER["SERVER_PROTOCOL"]	HTTP/1.1
_SERVER["REQUEST_SCHEME"]	http
_SERVER["GATEWAY_INTERFACE"]	CGI/1.1
_SERVER["SERVER_SOFTWARE"]	nginx/1.10.1
_SERVER["REMOTE_ADDR"]	192.168.78.1
_SERVER["REMOTE_PORT"]	58706
_SERVER["SERVER_ADDR"]	192.168.78.3
_SERVER["SERVER_PORT"]	80
_SERVER["SERVER_NAME"]	ng.test

图 5-9 环境变量信息

在 PHP 安装后，由于用户没有配置 php.ini，phpinfo 会在 Date 的相关信息下出现一个 Warning 警告信息，提示用户当前未对时区进行配置。读者可以在 php.ini 中找到[Date]下面的“;date.timezone＝配置项”，去掉分号注释，并设置为 UTC(协调世界时)、PRC(中国时区)或 Asia/Shanghai(亚洲上海时区)，然后重启 PHP-FPM 使配置生效后即可解决。

6. 判断 PHP 文件是否存在

在完成前面的步骤后，虽然能够正确执行 PHP 文件，但是还有一些细节问题需要注意，具体如下。

1) 404 页面问题

由于前面的配置将路径以.php 结尾的请求发送给 PHP，这就会导致如果用户请求的 PHP 文件不存在时，服务器返回 File not found.错误提示，而不是原来的 404 页面。

2) PATHINFO 问题

PATHINFO 是 Apache 支持的一种参数传递机制，它是指一个 URL 地址中，从一个有效的 PHP 脚本文件名到 URL 参数之间的部分。假设在网站目录下存在 test.php 文件，则各种 URL 的 PATHINFO 提取方式如表 5-3 所示。

表 5-3　PATHINFO 示例

URL 地址	PATHINFO
http://www.ng.test/test.php	无
http://www.ng.test/test.php?a=1	无
http://www.ng.test/test.php/abc	/abc
http://www.ng.test/test.php/abc/?a=1	/abc/
http://www.ng.test/test.php/1.php	/1.php(在 1.php 文件不存在的情况下)

PATHINFO 通常用于在某个脚本后面添加一些自定义内容(如/test.php/aa/bb)，网站利用这些内容可以进行 SEO 优化或增强用户体验。

需要注意的是，PHP 在 Apache 中是以模块方式工作的，而在 Nginx 环境下以 CGI 方式工作。在 php.ini 中，默认开启了 cgi.fix_pathinfo，用于在 CGI 模式下自动识别 PATHINFO，具体如下。

(1) 关闭时：cgi.fix_pathinfo＝0。

```
请求 URL: http://www.ng.test/test.php/a.php
执行文件: /usr/local/nginx/html/www.ng.test/test.php/a.php
文件不存在时,返回错误信息"No input file specified."并停止
```

(2) 开启时：cgi.fix_pathinfo＝1。

```
请求 URL: http://www.ng.test/test.php/a.php
执行文件: /usr/local/nginx/html/www.ng.test/test.php/a.php
文件不存在时,执行上级文件: /usr/local/nginx/html/www.ng.test/test.php
文件仍然不存在时,返回错误信息"File not found."并停止
```

从上述示例可以看出，在关闭 cgi.fix_pathinfo 的状态下，PHP 会严格按照给定的 URL 执行对应的 PHP 文件；若开启 cgi.fix_pathinfo，当 URL 对应的文件不存在时，PHP 会按照层级向上查找文件并执行。但若把示例中的 test.php 换成其他文件(如 pic.jpg)，如果 a.php 文件不存在，PHP 就会把 pic.jpg 当成 PHP 脚本执行，这显然会出现安全问题。

以上提到的 404 页面和 PATHINFO 两个问题，都是由于服务器中不存在请求的文件而导致的。为了防止这些情况带来的问题，可以利用 Nginx 提供的 try_files 指令来检测文件是否存在。接下来修改虚拟主机 www.ng.test 中关于 PHP 的 location 配置，具体命令如下。

```
location ~\.php$ {
    try_files $uri =404;
    fastcgi_pass 127.0.0.1:9000;
    include fastcgi.conf;
}
```

在上述配置中，try_files 指令的参数数量可以有多个，每个参数表示一个文件路径。try_files 会从左到右依次检测给定的文件路径是否存在，如果存在则发生内部重定向，跳出当前 location 并重新匹配。try_files 的最后一个参数可以是状态码，如=404 表示返回 404；如果最后一个参数是文件路径，当文件不存在时会返回 500 错误。$uri 是 Nginx 的内置变量，该变量的值是 URL 地址中从域名到参数之间的部分。

完成上述配置后，当用户试图访问一个以.php 结尾且文件不存在的路径时，Nginx 会优先返回 404 页面，并且不会发送给 PHP 执行。这样就同时解决了 404 页面和 PATHINFO 的问题。

5.2 Nginx+Apache 环境

在 Nginx 走进人们的视野之前，大量的网站都是基于 LAMP 平台运行的。在 Nginx 受到认可之后，人们发现可以将 Nginx 作为前端 Web 服务器，而 Apache 在后端只处理动态请求，实现动静态分离。这种方式既避免了更换 Web 服务器带来的不稳定性，又利用 Nginx 提升了性能，是一种便捷的优化方案。本节将讲解如何搭建 Nginx+Apache 环境。

5.2.1 Apache 的安装与使用

Apache HTTP Server 是一个功能强大的 Web 服务器，一直具有相当高的市场占有率。Apache 虽然在高并发性能上不如 Nginx，但是在功能上更加完善，具有非常出色的稳定性，目前仍然有大量的网站正在使用 LAMP 平台。而 Nginx 具有非常高的性能，两者可以共存，因此可以组成 LNAMP 平台，Nginx 处理静态请求，动态请求交给 Apache 和 PHP 进行处理。

1. 获取 Apache

在 Apache 的官方网站 https://httpd.apache.org 可以获取软件源代码的下载地址，如图 5-10 所示。值得一提的是，Apache 官方网站对 Apache HTTP Server 的简称是 httpd，

而国内更习惯简称为 Apache。本书也沿用国内习惯的 Apache 这一简称。

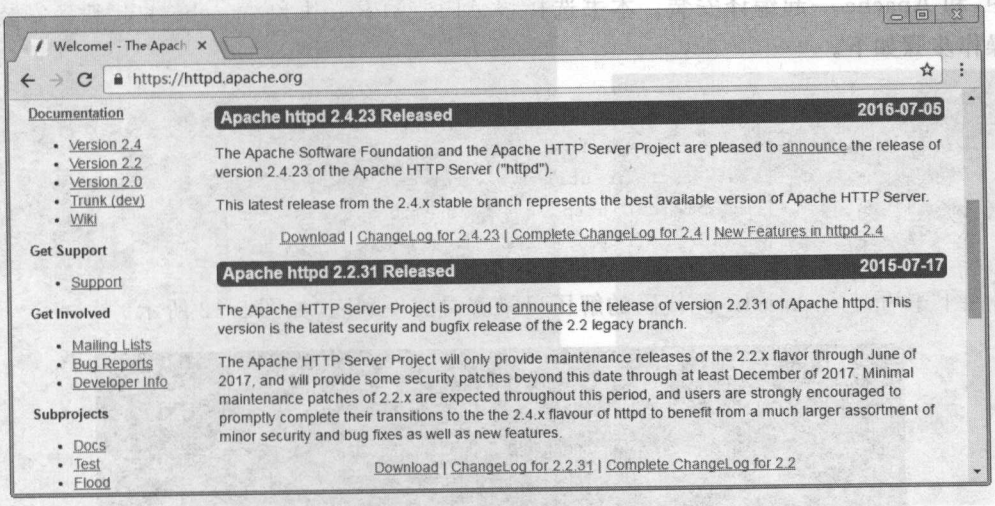


图 5-10　获取 Apache

从图 5-10 中可以看出，目前 Apache 发布了 2.2 和 2.4 两种版本，本书基于 2.4 版本进行讲解。在官方网站找到 Apache 2.4 的源代码包 httpd-2.4.23.tar.gz 进行下载即可。图片左上角的 Documentation 链接是 Apache 的官方文档，阅读文档可以查看 Apache 的详细使用说明。

需要注意的是，Apache 2.4 依赖于 Apache 的软件支持库 APR(Apache Portable Runtime)和 APR-util(Apache Portable Runtime Utility)，其下载地址为 https://apr.apache.org，如图 5-11 所示。

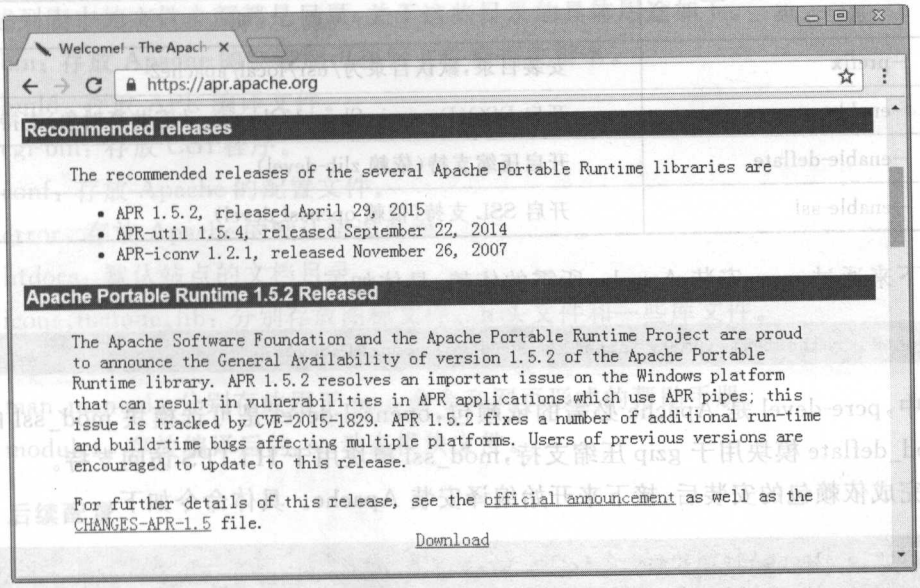


图 5-11　获取 APR 和 APR-util

将前面下载的 httpd-2.4.23.tar.gz、apr-1.5.2.tar.gz、apr-util-1.5.4.tar.gz 3 个源码

包保存到 Linux 服务器中。APR 和 APR-util 既可以独立编译安装，也可以放入 Apache 源码中，和 Apache 一起编译安装。本书选择将 APR、APR-util 和 Apache 一起编译安装，具体操作步骤如下。

```
[root@localhost ~]#tar -zxvf httpd-2.4.23.tar.gz
[root@localhost ~]#tar -zxvf apr-1.5.2.tar.gz
[root@localhost ~]#tar -zxvf apr-util-1.5.4.tar.gz
[root@localhost ~]#mv apr-1.5.2 httpd-2.4.23/srclib/apr
[root@localhost ~]#mv apr-util-1.5.4 httpd-2.4.23/srclib/apr-util
```

接下来可进入 Apache 源代码的解压目录查看文件列表，如图 5-12 所示。

```
[root@localhost ~]# cd httpd-2.4.23
[root@localhost httpd-2.4.23]# ls
ABOUT_APACHE     BuildBin.dsp    emacs-style     LAYOUT          NOTICE           srclib
acinclude.m4     buildconf       httpd.dep       libhttpd.dep    NWGNUmakefile    support
Apache-apr2.dsw  CHANGES         httpd.dsp       libhttpd.dsp    os               test
Apache.dsw       CMakeLists.txt  httpd.mak       libhttpd.mak    README           VERSIONING
apache_probes.d  config.layout   httpd.spec      LICENSE         README.cmake
ap.d             configure       include         Makefile.in     README.platforms
build            configure.in    INSTALL         Makefile.win    ROADMAP
BuildAll.dsp     docs            InstallBin.dsp  modules         server
[root@localhost httpd-2.4.23]#
```

图 5-12 Apache 解压后的文件列表

2. 编译安装 Apache

Apache 提供 configure 程序用于编译安装，使用./configure --help 命令可以查看详细的编译选项，也可以查阅 Apache 官方手册，其常用的编译选项如表 5-4 所示。

表 5-4 Apache2.4 常用编译选项

选　项	说　明
--prefix	安装目录，默认目录为/usr/local/apache2
--enable-so	开启 DSO(Dynamic Shared Objects，动态共享对象)支持
--enable-deflate	开启压缩支持(依赖 zlib-devel)
--enable-ssl	开启 SSL 支持(依赖 openssl-devel)

接下来通过 yum 安装 Apache 所需的依赖，具体如下。

```
[root@localhost ~]#yum -y install pcre-devel openssl-devel
```

其中，pcre-devel 是 Apache 必需的依赖包，openssl-devel 是可选模块 mod_ssl 的依赖包。mod_deflate 模块用于 gzip 压缩支持，mod_ssl 模块用于 HTTPS 访问支持。

在完成依赖包的安装后，接下来开始编译安装 Apache。具体命令如下。

```
[root@localhost ~]#cd httpd-2.4.23
[root@localhost httpd-2.4.23]#./configure --enable-so --enable-deflate --enable-ssl
[root@localhost httpd-2.4.23]#make && make install
```

成功安装后的结果如图 5-13 和图 5-14 所示。

```
creating test/Makefile
config.status: creating docs/conf/httpd.conf
config.status: creating docs/conf/extra/httpd-autoindex.conf
config.status: creating docs/conf/extra/httpd-dav.conf
config.status: creating docs/conf/extra/httpd-default.conf
config.status: creating docs/conf/extra/httpd-info.conf
config.status: creating docs/conf/extra/httpd-languages.conf
config.status: creating docs/conf/extra/httpd-manual.conf
config.status: creating docs/conf/extra/httpd-mpm.conf
config.status: creating docs/conf/extra/httpd-multilang-errordoc.conf
config.status: creating docs/conf/extra/httpd-ssl.conf
config.status: creating docs/conf/extra/httpd-userdir.conf
config.status: creating docs/conf/extra/httpd-vhosts.conf
config.status: creating docs/conf/extra/proxy-html.conf
config.status: creating include/ap_config_layout.h
config.status: creating support/apxs
config.status: creating support/apachectl
config.status: creating support/dbmmanage
config.status: creating support/envvars-std
config.status: creating support/log_server_status
config.status: creating support/logresolve.pl
config.status: creating support/phf_abuse_log.cgi
config.status: creating support/split-logfile
config.status: creating build/rules.mk
config.status: creating build/pkg/pkginfo
config.status: creating build/config_vars.sh
config.status: creating include/ap_config_auto.h
config.status: executing default commands
[root@localhost httpd-2.4.23]#
```

图 5-13　Apache 编译前的配置

```
Installing configuration files
mkdir /usr/local/apache2/conf
mkdir /usr/local/apache2/conf/extra
mkdir /usr/local/apache2/conf/original
mkdir /usr/local/apache2/conf/original/extra
Installing HTML documents
mkdir /usr/local/apache2/htdocs
Installing error documents
mkdir /usr/local/apache2/error
Installing icons
mkdir /usr/local/apache2/icons
mkdir /usr/local/apache2/logs
Installing CGIs
mkdir /usr/local/apache2/cgi-bin
Installing header files
Installing build system files
Installing man pages and online manual
mkdir /usr/local/apache2/man
mkdir /usr/local/apache2/man/man1
mkdir /usr/local/apache2/man/man8
mkdir /usr/local/apache2/manual
make[1]: Leaving directory `/root/httpd-2.4.23'
[root@localhost httpd-2.4.23]#
```

图 5-14　Apache 编译安装

在完成安装后，打开默认安装目录/usr/local/apache2 查看 Apache 的目录结构，具体如下。

```
[root@localhost ~]#cd /usr/local/apache2
[root@localhost apache2]#ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  lib  logs
man  manual  modules
```

上述列表中的文件全部都是目录，关于这些目录的具体用途如下。

- bin：存放 Apache 的二进制可执行文件和相关脚本。
- build：存放 APR 编译文件。
- cgi-bin：存放 CGI 程序。
- conf：存放 Apache 的配置文件。
- error：存放 Apache 的默认错误页面。
- htdocs：默认站点的文档目录。
- icons、include、lib：分别存放图标文件、.h 头文件和一些库文件。
- logs：存放日志文件，包括 access_log(访问日志)和 error_log(错误日志)。
- man、manual：分别存放用于 man 命令和网页形式的帮助手册。
- modules：存放编译后的.so 动态模块文件。

3. 后续配置

在安装 Apache 后，为了通过 service 命令管理 Apache 服务，在系统/etc/init.d 中编写脚本。

```
[root@localhost ~]#vi /etc/init.d/httpd
```

由于 Apache 在 bin 目录下已经提供了一个能够通过 start、stop、restart 等参数控制 Apache 服务的脚本程序 apachectl，因此在编写 httpd 服务脚本时直接将参数传入 apachectl 脚本中即可，具体如下。

```
#! /bin/bash
#chkconfig: 35 85 15
/usr/local/apache2/bin/apachectl $1
```

在保存 httpd 脚本文件后，为该文件添加可执行的权限。

```
[root@localhost ~]#chmod +x /etc/init.d/httpd
```

接下来通过 service httpd 命令实现 Apache 的启动、停止与重新启动，具体命令如下。

```
[root@localhost ~]#service httpd start
[root@localhost ~]#service httpd stop
[root@localhost ~]#service httpd restart
```

若之前已经安装 Nginx，则需要先停止 Nginx 服务后，再启动 Apache，以避免端口冲突。若是在全新的系统中安装 Apache，应该打开系统防火墙的 80 端口，使 Apache 能够被访问，具体命令如下。

```
[root@localhost ~]#iptables -I INPUT -p tcp --dport 80 -j ACCEPT
[root@localhost ~]#service iptables save
```

需要注意的是，Apache 在安装后默认没有配置 ServerName，因此在 Apache 启动时会出现一个提示信息，告知用户 Apache 自动使用了 localhost.localdomain 作为 ServerName。若不希望出现这个提示，可以打开 conf 目录下的 httpd.conf 文件，搜索找到如下一行配置，取消 # 注释即可。

```
#ServerName www.example.com:80
```

完成上述操作后，启动 Apache 服务并查看进程和监听端口，运行结果如图 5-15 所示。

```
[root@localhost conf]# ps aux | grep httpd
root     11760  0.0  0.5  70276  2632 ?        Ss   10:38   0:00 /usr/local/apache2/bin/httpd -k start
daemon   11862  0.0  0.4 414536  2148 ?        Sl   10:53   0:00 /usr/local/apache2/bin/httpd -k start
daemon   11863  0.0  0.4 414536  2152 ?        Sl   10:53   0:00 /usr/local/apache2/bin/httpd -k start
daemon   11864  0.0  0.4 414536  2148 ?        Sl   10:53   0:00 /usr/local/apache2/bin/httpd -k start
root     11955  0.0  0.1 103312   880 pts/1    S+   10:55   0:00 grep httpd
[root@localhost conf]# netstat -tlnp | grep httpd
tcp        0      0 :::80                       :::*                        LISTEN      11760/httpd
```

图 5-15 查看 Apache 进程和监听端口

从输出结果中可以看出，Apache 由 1 个 root 用户进程和 3 个 daemon 用户进程组成。另外，Apache 还具有 MPM（Multi-Processing Modules，多处理模块）机制，提供 event、prefork、worker 等多种 MPM 可以使用。通过如下命令可以查看当前安装的 Apache 正在使用哪一种 MPM 模块。

```
[root@localhost ~]#cd /usr/local/apache2/bin
```

```
[root@localhost bin]# ./httpd -M | grep mpm
mpm_event_module (static)
```

上述结果中的 mpm_event_module 表示 MPM 的 event 模块，后面的 static 表示此模块属于静态链接。Apache 的 MPM 模块可以在编译选项中通过--with-mpm 和--enable-mpms-shared 进行切换，具体可以阅读 configure 中的帮助说明。从实际使用的角度来说，worker 和 event 有较高的性能和灵活性，prefork 则具有较高的可靠性和兼容性。默认情况下 Apache 会自动选择适合当前系统的 MPM 模块。

4. 访问测试

在浏览器中访问 Linux 服务器，测试 Apache 是否已经可以使用。如果出现如图 5-16 所示的页面，则表示 Apache 当前正在运行。

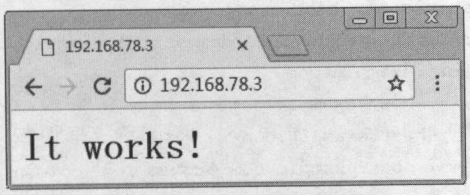


图 5-16　访问测试

5.2.2　Apache 的基本配置

Apache 和 Nginx 同是 Web 服务器，因此在功能上有许多相似的地方。Apache 的配置文件 httpd.conf 位于 conf 目录，该配置文件的语法与 Nginx 不同。下面介绍 Apache 的基本配置。

1. 基本指令

在 httpd.conf 配置文件中有一些基本指令，用于配置网站目录、端口号、域名等，如表 5-5 所示。

表 5-5　Apache 基本配置指令

指　令	说　明
ServerRoot	Apache 服务器的安装目录
Listen	服务器监听的端口号，如 80、443
LoadModule	需要加载的模块
ServerAdmin	服务器管理员的邮箱地址
ServerName	服务器的域名
DocumentRoot	网站根目录
ErrorLog	记录错误日志

上述配置读者可根据需要进行修改，若因修改错误造成 Apache 无法启动，可还原

conf/original目录中的配置文件备份。

2. 配置虚拟主机

Apache的虚拟主机可以在conf/httpd.conf文件中配置，也可以在conf/extra/httpd-vhosts.conf中配置。后者是在httpd.conf文件中通过Include指令引入的子配置文件，但是在使用前需要先在httpd.conf中找到如下一行配置取消注释，否则httpd-vhosts.conf将不会生效。

```
#Include conf/extra/httpd-vhosts.conf
```

接下来打开httpd-vhosts.conf文件，可以看到Apache提供的默认配置，具体如下。

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot "/usr/local/apache2/docs/dummy-host.example.com"
    ServerName dummy-host.example.com
    ServerAlias www.dummy-host.example.com
    ErrorLog "logs/dummy-host.example.com-error_log"
    CustomLog "logs/dummy-host.example.com-access_log" common
</VirtualHost>
```

上述配置中，第1行的*.80表示该主机通过80端口访问；ServerAdmin是管理员邮箱地址；DocumentRoot是该虚拟主机的文档目录；ServerName是虚拟主机的域名；ServerAlias用于配置多个域名别名(用空格分隔)，支持形如*.example.com的泛解析二级域名；ErrorLog是错误日志；CustomLog是访问日志，其后的common表示日志格式为通用格式。

由于上述示例配置并不需要使用，所以加上#注释即可。下面实现手动配置一个虚拟主机，省略管理员邮箱，不记录日志。具体命令如下。

```
<VirtualHost *:80>
    DocumentRoot "htdocs/www.ng.test"
    ServerName www.ng.test
    ServerAlias ng.test
</VirtualHost>
```

保存配置文件后，创建DocumentRoot指令中指定的文档目录，然后重启Apache服务使配置生效。为了测试，在文档目录中编写index.html，文件内容是Welcome www.ng.test，具体操作如下。

```
[root@localhost ~]#mkdir /usr/local/apache2/htdocs/www.ng.test
[root@localhost ~]#service httpd restart
[root@localhost ~]#cd /usr/local/apache2/htdocs/www.ng.test
[root@localhost www.ng.test]#echo 'Welcome www.ng.test' >index.html
```

接下来通过浏览器进行访问，确保物理机hosts文件中www.ng.test和ng.test两个域名都解析到Linux服务器的IP地址上，成功后的显示结果如图5-17所示。

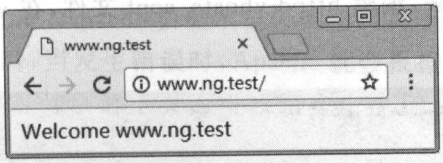


图 5-17　测试虚拟主机

3. 访问权限控制

Apache 可以控制服务器中的哪些目录允许被外部访问。在 httpd.conf 中，默认站点目录 htdocs 已经配置为允许外部访问，但如果要将其他目录也允许访问时，需要手动进行配置。下面通过虚拟主机 www.test.com 来介绍如何进行访问权限控制。

编辑 httpd-vhosts.conf，在配置虚拟主机的同时，通过 Require 指令控制访问权限，具体如下。

```
<VirtualHost *:80>
    DocumentRoot "/var/www/www.test.com"
    ServerName www.test.com
</VirtualHost>
<Directory "/var/www/www.test.com">
    Require all granted
</Directory>
```

上述配置，将虚拟主机的站点目录指定到/var/www/www.test.com 目录下，并通过<Directory>指令为其配置了目录访问权限。其中，Require all granted 表示允许所有的访问，若只允许本地访问可设为 Require local，拒绝所有访问可设为 Require all denied。

接下来创建虚拟主机的文档目录和测试文件，具体步骤如下。

```
[root@localhost ~]#mkdir -p /var/www/www.test.com
[root@localhost ~]#cd /var/www/www.test.com
[root@localhost www.test.com]#echo Welcome www.test.com >index.html
```

在完成相关文件的创建后，重启 Apache 服务使配置生效，然后通过浏览器访问。当用户有访问权限时，就会出现如图 5-18 所示的结果。如果没有权限访问时，就会出现 Forbidden 错误提示。

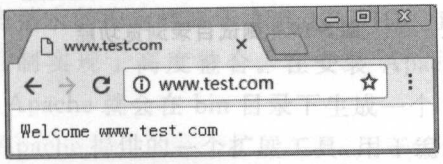


图 5-18　测试访问权限

4. 分布式配置文件

分布式配置文件是为目录单独进行配置的文件，可以实现在不重启服务器的前提下更

改某个目录的配置。接下来，编辑 httpd-vhosts.conf 文件，在 www.test.com 目录配置中开启分布式配置文件。

```
<Directory "/var/www/www.test.com">
    Require all granted
    AllowOverride All
</Directory>
```

当上述配置添加 AllowOverride All 之后，Apache 就会到站点下的各个目录中读取分布式配置文件，文件名为.htaccess(“.”表示隐藏文件，而不是文件扩展名)，该文件中的配置将会覆盖原有的目录配置。在分布式配置文件中可以直接编写<Directory>中的大部分配置，如通过 Options 指令开启或关闭目录浏览，通过 ErrorDocument 指令自定义错误页面。

Apache 分布式配置文件虽然方便了网站管理员对目录的管理，但是会影响服务器的运行效率。因此，需要将其关闭时，改为 AllowOverride None 即可。

5. 目录浏览功能

当开启 Apache 目录浏览功能时，如果访问的目录中没有默认索引页(如 index.html)，就会显示目录中的文件列表。下面在目录/var/www/www.test.com 中创建.htaccess 文件，编写如下配置。

```
DirectoryIndex index.html index.htm
Options Indexes
```

上述配置中，DirectoryIndex 指令用于配置默认索引页，Options 指令用于配置目录选项，Indexes 表示启用文件列表。当配置生效后，文件列表的显示效果如图 5-19 所示。另外，若要关闭目录浏览功能时，将其修改为 Options -Indexes 即可。

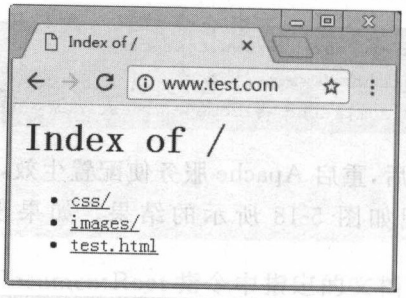


图 5-19 测试目录浏览功能

6. 自定义错误页面

Apache 通过 ErrorDocument 指令配置错误页面，示例配置如下。

```
ErrorDocument 403 /403.html
ErrorDocument 404 /404.html
ErrorDocument 500 /500.html
```

在自定义错误页面之前，Apache 会显示自带的错误页面，HTML 模板保存在 Apache 的 error 目录。而在配置后，当发生错误时，Apache 就会根据 ErrorDocument 指令配置的状态码显示对应的自定义页面，指定页面可以是一个 URL 地址，或站点目录下的某个文件。

7. 模块开启和关闭

Apache 支持将一些模块编译成独立的. so 文件保存在 modules 目录中，然后利用 httpd. conf 中的 LoadModule 指令开启或关闭这些模块，这是一种非常灵活的动态模块机制。Nginx 从 1. 9. 11 版本开始也增加了类似的动态模块机制。

在前面的安装步骤中，安装 Apache 时通过编译选项选择 mod_deflate、mod_ssl 两个附加模块，这两个模块编译后的. so 文件就保存在 modules 目录，如下所示。

```
[root@localhost apache2]#ls modules | grep -P 'deflate|ssl'
mod_deflate.so
mod_ssl.so
```

在 Apache 安装后上述这两个模块没有被开启，需要在 httpd. conf 中手动开启。编辑 httpd. conf 文件，分别找到如下两行配置，取消 # 注释，然后保存文件并重启 Apache 服务即可使模块生效。

```
#LoadModule deflate_module modules/mod_deflate.so
#LoadModule ssl_module modules/mod_ssl.so
```

为了测试模块是否开启，可以使用 apachectl 命令的-M 选项显示已开启的模块列表，运行结果如下。

```
[root@localhost apache2]#./bin/apachectl -M | grep -P 'deflate|ssl'
deflate_module(shared)
ssl_module(shared)
```

在上述输出结果中，shared 表示该模块属于动态模块，和 static 静态模块相比，动态模块可以自由开启和关闭，但相比静态模块在性能方面有所损失。

5.2.3　Apache 与 PHP 整合

Apache 处理 PHP 动态请求的稳定性高于 Nginx+PHP-FPM 的方式，这是因为 PHP 利用 Apache 的动态模块机制实现了高度整合。在安装 Apache 时，编译选项中有一个 --enable-so，添加此选项后，Apache 就会在 bin 目录下生成一个 apxs 程序。apxs 的含义为 Apache extension tool，是 Apache 提供的一个扩展工具，用于编译模块。PHP 的 configure 程序可以利用 apxs 编译一个用于 Apache 访问 PHP 的模块，以实现整合。下面对这种方式进行详细讲解。

1. 编译安装 PHP

在前面讲解的 PHP 安装步骤中，生成了 PHP 的 FastCGI 进程管理器 PHP-FPM，而在

和Apache整合时无须这个管理器。PHP提供编译选项--with-apxs2用于编译Apache模块，其中数字2表示该编译选项是为2.x版本的Apache设计的。将该选项指向Apache的bin目录下的apxs程序，具体操作如下。

```
[root@localhost ~]#cd php-5.6.27
[root@localhost php-5.6.27]#make clean     #若之前安装过Nginx+PHP,则需要清理
[root@localhost php-5.6.27]#./configure --prefix=/usr/local/php \
--with-apxs2=/usr/local/apache2/bin/apxs --with-zlib --enable-zip \
--enable-mbstring --with-mcrypt --with-mysql --with-mysqli --with-pdo-mysql \
--with-gd --with-jpeg-dir --with-png-dir --with-freetype-dir --with-curl \
--with-openssl --with-mhash --enable-bcmath --enable-opcache
[root@localhost php-5.6.27]#make && make install
```

完成PHP的编译安装后，在Apache的modules目录中可以看到编译后的PHP模块，在httpd.conf配置文件中会看到该模块已经添加并启用。

```
[root@localhost ~]#cd /usr/local/apache2
[root@localhost apache2]#ls modules | grep php
libphp5.so
[root@localhost apache2]#grep php conf/httpd.conf
LoadModule php5_module        modules/libphp5.so
```

若初次安装PHP，还需要将php.ini复制到php的lib目录中。

```
[root@localhost apache2]#cd ~/php-5.6.27
[root@localhost php-5.6.27]#cp php.ini-development /usr/local/php/lib/php.ini
```

值得一提的是，php.ini的保存目录既可以使用--with-config-file-path编译选项来指定，也可以在httpd.conf文件中通过PHPIniDir指令进行指定。

2. Apache相关配置

为了使Apache能够识别php扩展名的文件，将文件交给PHP模块处理，需要在httpd.conf中配置文件的扩展名。编辑httpd.conf文件，添加如下配置即可。

```
<FilesMatch "\.php$">
    setHandler application/x-httpd-php
</FilesMatch>
```

除了文件扩展名，还需要配置目录索引页，使目录中的index.php文件可作为默认页面。在httpd.conf中搜索DirectoryIndex，找到如下配置，将index.php添加到index.html后面即可。

```
<IfModule dir_module>
    DirectoryIndex index.html index.php
</IfModule>
```

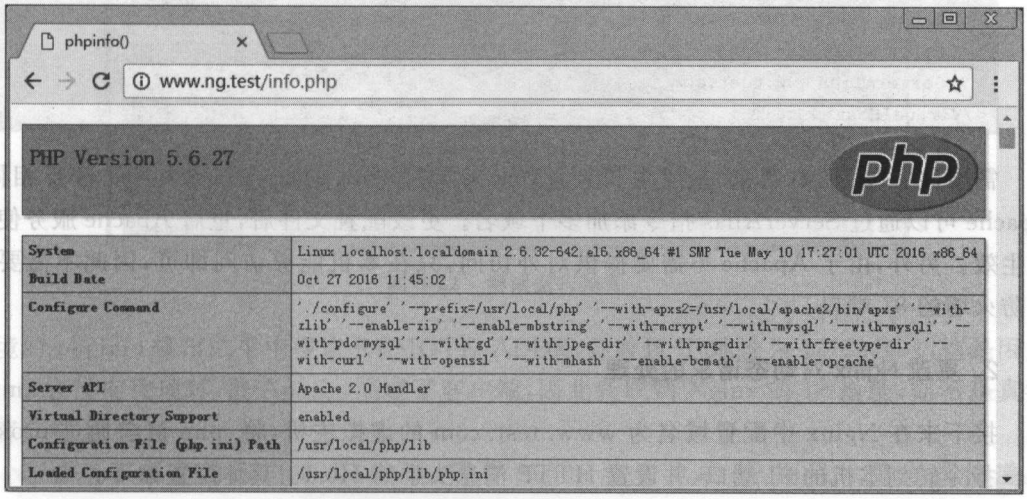
3. 访问测试

为了测试Apache+PHP环境是否正确安装和配置成功，接下来在www.ng.test文档

目录下创建文件 info.php，编写代码输出 phpinfo 信息，具体操作如下。

```
[root@localhost ~]#cd /usr/local/apache2/htdocs/www.ng.test/
[root@localhost www.ng.test]#echo '<?php phpinfo();' >info.php
```

可通过浏览器访问 http://www.ng.test/info.php，如图 5-20 所示。



System	Linux localhost.localdomain 2.6.32-642.el6.x86_64 #1 SMP Tue May 10 17:27:01 UTC 2016 x86_64
Build Date	Oct 27 2016 11:45:02
Configure Command	'./configure' '--prefix=/usr/local/php' '--with-apxs2=/usr/local/apache2/bin/apxs' '--with-zlib' '--enable-zip' '--enable-mbstring' '--with-mcrypt' '--with-mysql' '--with-mysqli' '--with-pdo-mysql' '--with-gd' '--with-jpeg-dir' '--with-png-dir' '--with-freetype-dir' '--with-curl' '--with-openssl' '--with-mhash' '--enable-bcmath' '--enable-opcache'
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	/usr/local/php/lib
Loaded Configuration File	/usr/local/php/lib/php.ini

图 5-20　Apache 环境 phpinfo 输出结果

5.2.4　Nginx＋Apache 动静分离

在 Web 服务器中，动态请求是指该请求需要服务器端的程序处理。例如，当用户请求一个 PHP 脚本文件时，就会调用 PHP 处理，并返回该脚本的处理结果。而静态请求不需要程序处理，直接读取文件并返回即可，如 HTML、CSS、JavaScript、图片等文件。针对这两种请求各自的特点，可以由 Nginx 提供对外访问，静态请求直接由 Nginx 处理，动态请求转交给 Apache 处理，这样就实现了动静分离，如图 5-21 所示。

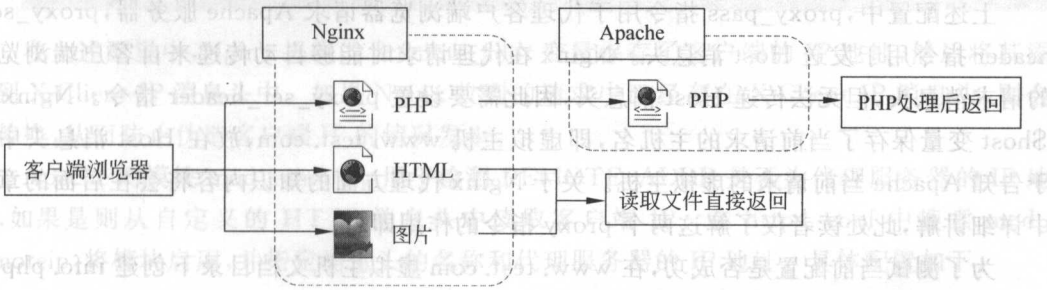


图 5-21　Nginx＋Apache 动静分离

1. 更改 Apache 监听端口

当 Nginx 作为前端 Web 服务器后，就会占用 80 端口。而为了让 Apache 正常运行，需要改变 Apache 监听的端口号。下面更改 Apache 的 httpd.conf 文件，将端口号改为 81。

```
Listen 81
```

除上述配置外，Apache虚拟主机 httpd-vhosts. conf 配置文件中的端口号也需要进行修改。

```
<VirtualHost *:81>
    DocumentRoot "/var/www/www.test.com"
    ServerName test.com
    ServerAlias www.test.com
</VirtualHost>
```

需要注意的是，在配置虚拟主机时，Apache 与 Nginx 的站点目录、主机名要相同。Apache 可以通过 ServerAlias 指令添加多个域名。更改配置文件后，重启 Apache 服务使配置生效。另外，由于 Apache 不需要提供对外访问，只在本机能够访问即可，因此不需要打开防火墙的 81 端口。

2. 更改 Nginx 对动态请求的处理

接下来在 Nginx 中配置域名为 www. test. com 的虚拟主机，将. php 请求通过 proxy_pass 指令转到本机的 81 端口，并设置 HTTP 消息头中的 Host。具体配置如下。

```
server {
    listen 80;
    server_name test.com www.test.com;
    root /var/www/www.test.com;
    index index.html index.htm index.php;
    location ~\.php${
        proxy_pass http://127.0.0.1:81;
        proxy_set_header Host $host;
    }
}
```

上述配置中，proxy_pass 指令用于代理客户端浏览器请求 Apache 服务器，proxy_set_header 指令用于发送 Host 消息头。Nginx 在代理请求时能够自动传递来自客户端浏览器的请求消息头，但无法传递 Host 消息头，因此需要设置 proxy_set_header 指令。Nginx 的 $host 变量保存了当前请求的主机名，即虚拟主机 www. test. com，放在 Host 消息头中用于告知 Apache 当前请求的虚拟主机。关于 Nginx 代理方面的知识内容将会在后面的章节中详细讲解，此处读者仅了解这两个 proxy 指令的作用即可。

为了测试当前配置是否成功，在 www. test. com 虚拟主机文档目录下创建 info. php 文件，编写代码输出 phpinfo 信息。运行结果如图 5-22 所示，可以看到 PHP 显示的 Server API 为 Apache，说明 PHP 正在运行于 Apache 服务器。能够看到这个结果，说明 Nginx 在服务器端代理请求了 Apache 服务器。

3. 传递客户端 IP 地址

在 Web 服务器中，REMOTE_ADDR 表示客户端的 IP 地址，而如果在 Apache+PHP

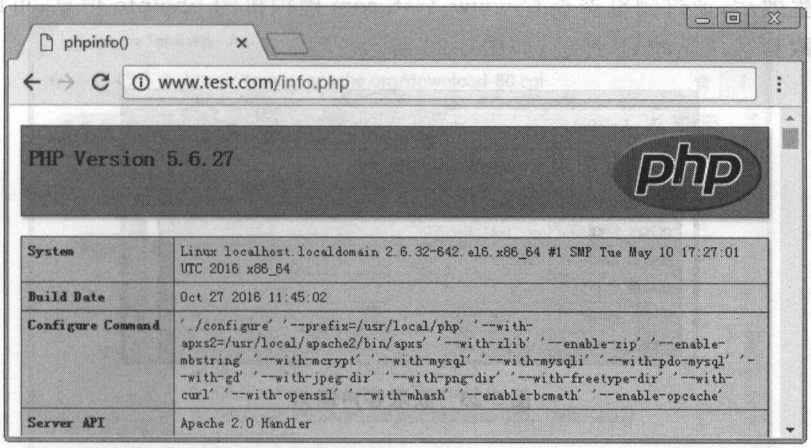


图 5-22　测试运行结果

环境的 phpinfo 输出结果中搜索 REMOTE_ADDR，会看到 IP 地址为 127.0.0.1，这是因为 Nginx 在反向代理时，被 Apache 当成了客户端，因此获取到 Nginx 的 IP 地址，而不是真实的客户端 IP 地址。

若要获取真实的客户端 IP 地址，需要 Nginx 将客户端 IP 用另一种方式传递给 Apache，这就要借助 HTTP 请求消息头。我们可以定义一个专门用于传递客户端 IP 的消息头 X-Client-IP，当 Nginx 获取到客户端 IP 后，将 IP 放入该消息头中发送给 Apache，然后在 Apache 中替换 REMOTE_ADDR 即可。

接下来在 Nginx 的 www.test.com 虚拟主机原有配置的基础上增加 X-Client-IP 消息头，具体如下。

```
location ~\.php${
    proxy_pass http://127.0.0.1:81;
    proxy_set_header Host $host;
    proxy_set_header X-Client-IP $remote_addr;
}
```

在上述配置中，Nginx 提供的 $remote_addr 变量保存了客户端的 IP 地址，然后将其添加到 X-Client-IP 消息头中。如果 Nginx 收到的请求中已经存在 X-Client-IP 消息头，则会被替换，从而防止伪造客户端 IP 的情况发生。

Apache 的模块 mod_remoteip 用于检测 REMOTE_ADDR 是否为代理服务器的 IP 地址，如果是则从自定义的 HTTP 消息头中读取客户端 IP。在 httpd.conf 中搜索 mod_remoteip，将模块启用，并指定消息头的名称和代理服务器的 IP 地址。具体配置如下。

```
LoadModule remoteip_module modules/mod_remoteip.so
RemoteIPHeader X-Client-IP
RemoteIPInternalProxy 127.0.0.1
```

完成上述配置后，当 Apache 的 mod_remoteip 模块检测到 REMOTE_ADDR 为 127.0.0.1 时，就会用 X-Client-IP 的值替换 REMOTE_ADDR 的值，然后清除 X-Client-IP 消息头。

在配置生效后，通过浏览器查看 www.test.com 虚拟机的 phpinfo 信息，可以看到当前已经正确获取到客户端的 IP 地址，如图 5-23 所示。

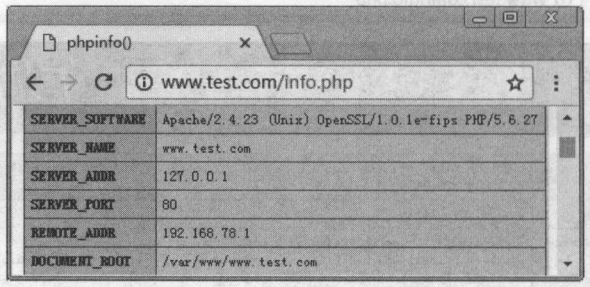


图 5-23 获取客户端 IP

5.3 Nginx＋Tomcat 环境

Nginx 处理静态资源的优势同样可以应用在 Tomcat 环境中。从实现方法上来说，Nginx＋Tomcat 环境的搭建思路与前面完成的 Nginx＋Apache 环境是完全相同的，只需要将 Nginx 与 Tomcat 的站点文档目录配置到同一目录下，利用 Nginx 的 proxy_pass 指令代理请求动态文件即可。本节将针对 Nginx＋Tomcat 环境进行详细讲解。

5.3.1 Tomcat 的安装与使用

1. 获取 Tomcat 和 JRE

由于 Tomcat 主要用于运行 JavaWeb 项目，所以需要在系统中安装 JRE(Java Runtime Environment，Java 运行环境)。如果需要直接在服务器中开发 JavaWeb 项目，则需要安装 JDK(Java Development Kit，Java 开发工具包)。本书选择只安装 JRE，在 Java 语言的官方网站 https://www.java.com 可以获取 JRE 的下载地址，如图 5-24 所示。

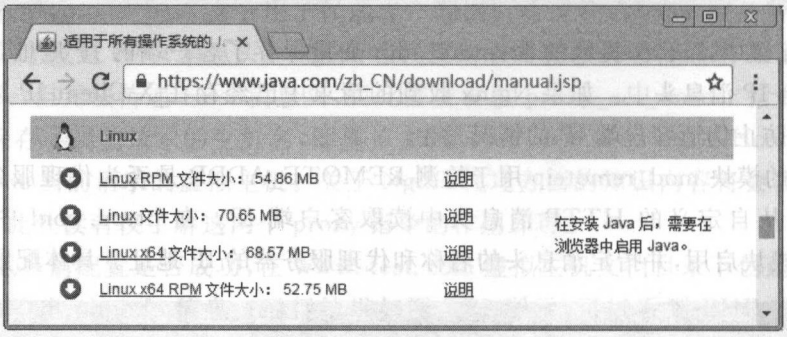


图 5-24 获取客户端 IP

接下来到 Apache 的官方网站中获取 Tomcat 的下载地址，Tomcat 的主页为 https://tomcat.apache.org，目前 Tomcat 发布了 6.x、7.x、8.x、9.x 4 种版本，本书选择 8.5 版，如图 5-25 所示。

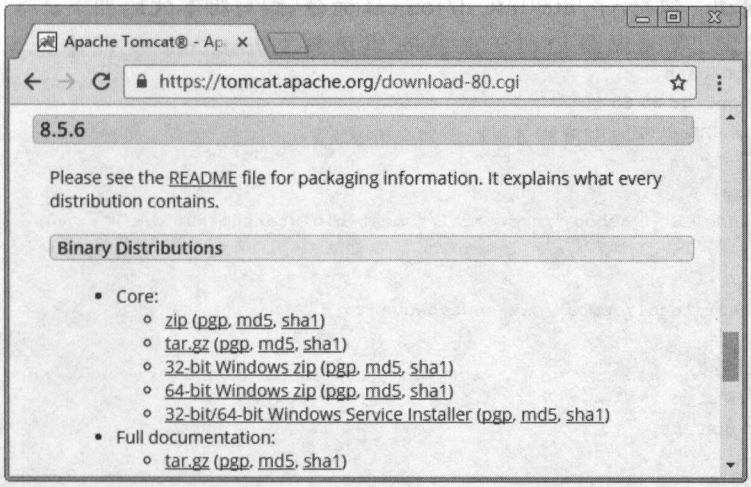


图 5-25　获取 Tomcat

在获取 JRE(jre-8u111-linux-x64. tar. gz)和 Tomcat(apache-tomcat-8. 5. 6. tar. gz)后，保存到 Linux 服务器中，然后按照如下操作步骤进行安装。

```
[root@localhost ~]#tar -zxvf jre-8u111-linux-x64.tar.gz
[root@localhost ~]#tar -zxvf apache-tomcat-8.5.6.tar.gz
[root@localhost ~]#mv jre1.8.0_111 /usr/local/jre
[root@localhost ~]#mv apache-tomcat-8.5.6 /usr/local/tomcat
```

从上述步骤可以看出，JRE 和 Tomcat 的安装非常简单，只需解压文件后移动到/usr/local 目录中即可。这是因为官方发布的是编译后的包，无须手动编译安装。

2. 查看 Tomcat 目录结构

切换到 Tomcat 目录中查看目录结构，具体结果如下。

```
[root@localhost tomcat]#ls
bin  conf  lib  LICENSE  logs  NOTICE  RELEASE-NOTES  RUNNING.txt  temp
webapps  work
```

关于 Tomcat 目录结构的具体作用如下。

- bin：存放可执行文件和脚本。
- conf：存放各种配置文件，如 web. xml、server. xml。
- lib：存放一些扩展名为 jar 的库文件。
- logs：存放日志文件。
- temp：存放 Tomcat 运行时产生的临时文件。
- webapps：Web 应用程序的主要发布目录。
- work：Tomcat 的工作目录，存放 JSP 编译生成的 Servlet 源文件和字节码文件。

3. 将 Tomcat 添加为系统服务

接下来编写 Tomcat 的服务脚本，实现通过 service 命令启动、关闭或重启 Tomcat 服

务，以及开机启动。执行 vi /etc/init.d/tomcat 命令，编写脚本代码如下。

```
1   #!/bin/bash
2   #chkconfig: 35 85 15
3   export JRE_HOME=/usr/local/jre
4   case "$1" in
5     start)
6       sudo -E -u nobody /usr/local/tomcat/bin/startup.sh
7     ;;
8     stop)
9       /usr/local/tomcat/bin/shutdown.sh
10    ;;
11    restart)
12      $0 stop
13      $0 start
14    ;;
15  esac
```

在上述代码中，第 3 行用于配置 JRE 的环境变量供 Tomcat 使用；第 6 行和第 9 行执行 Tomcat 的 bin 目录下的脚本实现服务的启动与关闭，其中 sudo -E -u nobody 表示以 nobody 用户身份运行并维持当前设置的环境变量；第 12～13 行的 $0 表示调用脚本自身。

使用 nobody 用户身份运行 Tomcat，可以避免直接用 root 身份运行所带来的安全风险（即最小权限原则）。为了让 nobody 用户能够获得 Tomcat 目录的操作权限，需要设置目录的所属用户和组。

```
[root@localhost ~]#chown -R nobody:nobody /usr/local/tomcat
```

完成上述操作后，为 tomcat 服务脚本文件设置执行权限，设置开机启动，然后启动服务。

```
[root@localhost ~]#chmod +x /etc/init.d/tomcat
[root@localhost ~]#chkconfig --add tomcat
[root@localhost ~]#service tomcat start
Using CATALINA_BASE:       /usr/local/tomcat
Using CATALINA_HOME:       /usr/local/tomcat
Using CATALINA_TMPDIR:     /usr/local/tomcat/temp
Using JRE_HOME:            /usr/local/jre
Using CLASSPATH:           /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/
tomcat-juli.jar
```

在 Tomcat 启动后，查看 Tomcat 的进程和监听的端口，具体如图 5-26 所示。

```
[root@localhost ~]# ps aux | grep tomcat
nobody   20809  4.5 13.3 2561816 67128 pts/0   Sl   03:48   0:03 /usr/local/jre/bin/java -Djava.util
.logging.config.file=/usr/local/tomcat/conf/logging.properties -Djava.util.logging.manager=org.apach
e.juli.ClassLoaderLogManager -Djdk.tls.ephemeralDHKeySize=2048 -Djava.protocol.handler.pkgs=org.apac
he.catalina.webresources -classpath /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat
-juli.jar -Dcatalina.base=/usr/local/tomcat -Dcatalina.home=/usr/local/tomcat -Djava.io.tmpdir=/usr/
local/tomcat/temp org.apache.catalina.startup.Bootstrap start
root     20845  0.0  0.1 103312   880 pts/0    S+   03:50   0:00 grep tomcat
[root@localhost ~]# netstat -tnlp | grep java
tcp        0      0 ::ffff:127.0.0.1:8005       :::*                    LISTEN      20809/java
tcp        0      0 :::8009                     :::*                    LISTEN      20809/java
tcp        0      0 :::8080                     :::*                    LISTEN      20809/java
```

图 5-26 查看 Tomcat 进程和监听端口

Tomcat 启动后默认会监听 3 个端口，分别是 8005、8009 和 8080。通过查看 Tomcat 的 conf 配置文件目录中的 server. xml 文件可以查看关于端口号的配置，其关键内容如下所示。

```
<Server port="8005" shutdown="SHUTDOWN">
  ⋮
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
                   redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
      ⋮
    </Engine>
  </Service>
</Server>
```

从上述配置可以看出，8005 端口用于关闭 Tomcat，8080 端口用于提供对外的 Web 访问(HTTP/1.1 协议)，8009 端口用于与其他 Web 服务器建立 AJP(Apache JServ Protocol)协议连接。在<Engine>标签中配置了默认主机 localhost，站点目录为 webapps。

4. 访问测试

由于 Tomcat 默认监听 8080 端口，为了能够通过客户端浏览器进行访问，需要在防火墙中打开 8080 端口，具体操作如下。

```
[root@localhost ~]#iptables -I INPUT -p tcp --dport 8080 -j ACCEPT
[root@localhost ~]#service iptables save
```

使用浏览器访问 http://192.168.78.3:8080 的结果如图 5-27 所示。

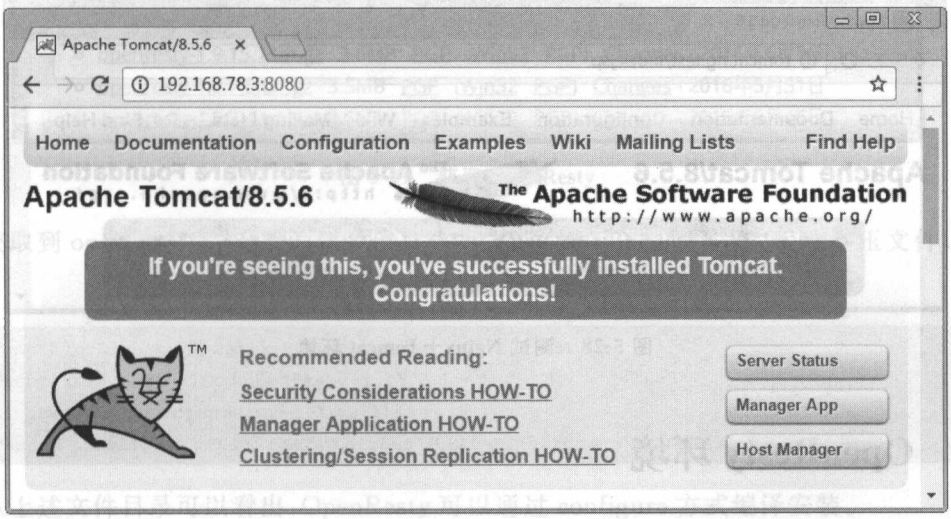


图 5-27　访问 Tomcat 默认站点

5.3.2 Nginx＋Tomcat 动静分离

JavaWeb 项目中的动态资源使用 jsp 或 do 扩展名，并且在站点文档目录中通过 WEB-INF 和 META-INF 目录保存一些配置信息。下面以 Tomcat 的默认站点为例，实现 Nginx＋Tomcat 的动静分离。在 Nginx 中创建虚拟主机 tomcat.ng.test，具体配置如下。

```
1   server {
2       listen 80;
3       server_name tomcat.ng.test;
4       root /usr/local/tomcat/webapps/ROOT;
5       index index.html index.htm index.jsp index.do;
6       location ~/(WEB-INF|META-INF){
7           deny all;
8       }
9       location ~\.(jsp|do)${
10          proxy_pass http://127.0.0.1:8080;
11          proxy_set_header X-Client-IP $remote_addr;
12      }
13      location ~^/(docs|examples)(/.*)*${
14        root /usr/local/tomcat/webapps;
15      }
16  }
```

在上述配置中，第 4 行设置站点文档目录为 webapps 目录下的 ROOT 目录，该目录是站点的根目录；第 5 行在默认页面配置中增加了 jsp 和 do 扩展名；第 6 行用于阻止访问配置目录；第 9 行将带有 jsp 和 do 扩展名的请求代理到 8080 端口的 Tomcat 服务器中；第 14 行配置了站点下 docs、examples 两个目录的实际存放路径。

接下来在物理机的 hosts 文件中将域名 tomcat.ng.test 解析到 192.168.78.3，然后使用浏览器访问进行测试，运行结果如图 5-28 所示。

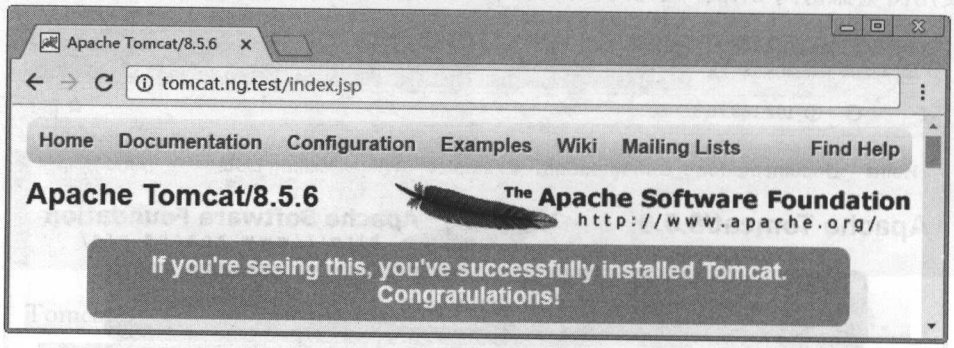


图 5-28 测试 Nginx＋Tomcat 环境

5.4 OpenResty 环境

Nginx 虽然拥有卓越的性能，但是在处理复杂需求时仍然需要借助 PHP、Python、JVM 等语言程序，这些语言相较 Nginx 基于 C 语言的模块具有学习门槛低、开发速度快的优势，

但并不适用于一些对性能要求极高的场合。OpenResty 就是一个为了兼顾开发效率和高性能而诞生的 Web 平台，它整合了 Nginx 与轻量级的 Lua 脚本语言，支持数据库的访问。本节将详细介绍 OpenResty 环境。

5.4.1　OpenResty 的安装与使用

OpenResty 是一个基于 Nginx 与 Lua 的高性能 Web 平台，其内部集成大量精良的 Lua 库、第三方模块，用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。

OpenResty 实现了将 Web 服务直接整合在 Nginx 服务的内部，从而充分利用 Nginx 的非阻塞 I/O 模型提高性能。OpenResty 不仅针对于 HTTP 客户端请求，甚至对于远程后端诸如 MySQL、PostgreSQL、Memcached 以及 Redis 等数据库系统都能一致的高性能响应。

1. 获取 OpenResty

通过 OpenResty 的官方网站 https://openresty.org 可以获取软件源代码的下载，如图 5-29 所示。

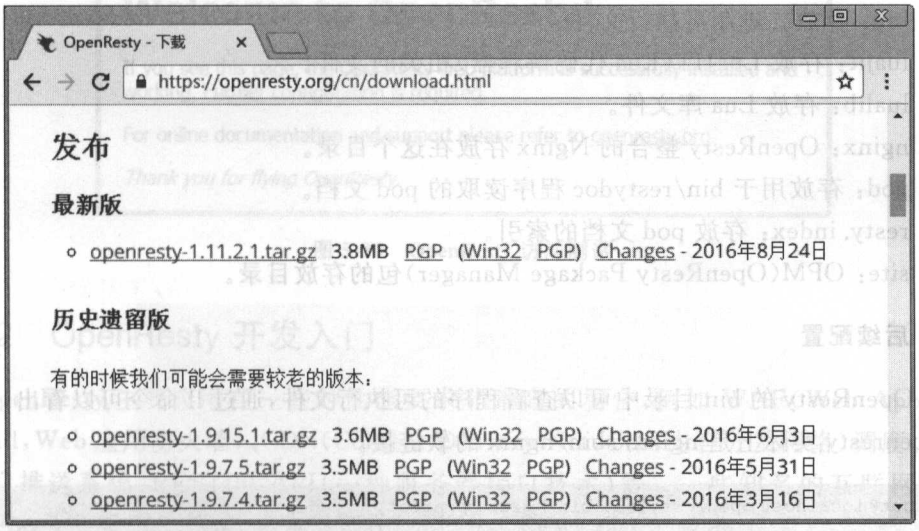


图 5-29　获取 OpenResty

获取到 openresty-1.11.2.1.tar.gz 文件后，保存到 Linux 服务器，然后解压文件，查看目录。

```
[root@localhost ~]#tar -zxvf openresty-1.11.2.1.tar.gz
[root@localhost ~]#cd openresty-1.11.2.1
[root@localhost openresty-1.11.2.1]#ls
bundle  configure  COPYRIGHT  patches  README.markdown  README-win32.txt  util
```

从上述文件目录可以看出，OpenResty 可以通过 configure 方式编译安装。

2. 编译安装 OpenResty

在编译安装 OpenResty 之前，需要安装所需的依赖包，具体如下。

```
[root@localhost ~]#yum -y install perl pcre-devel openssl-devel
```

完成上述依赖包的安装后，通过./configure --help 命令可以查看编译选项说明，其中--prefix 用于指定安装目录，默认为/usr/local/openresty。具体安装步骤如下。

```
[root@localhost openresty-1.11.2.1]#./configure
[root@localhost openresty-1.11.2.1]#make && make install
```

完成编译安装后，切换到 OpenResty 安装目录查看目录结构。

```
[root@localhost ~]#cd /usr/local/openresty
[root@localhost openresty]#ls
bin  luajit  lualib  nginx  pod  resty.index  site
```

关于 OpenResty 目录结构的具体说明如下。

- bin：存放二进制可执行文件。
- luajit：存放 LuaJIT(Lua 代码解释器)相关的文件。
- lualib：存放 Lua 库文件。
- nginx：OpenResty 整合的 Nginx 存放在这个目录。
- pod：存放用于 bin/restydoc 程序读取的 pod 文档。
- resty.index：存放 pod 文档的索引。
- site：OPM(OpenResty Package Manager)包的存放目录。

3. 后续配置

在 OpenResty 的 bin 目录中可以查看程序的可执行文件，通过 ll 命令可以看出，可执行文件 openresty 实际上是 nginx/sbin/nginx 的软链接。

```
[root@localhost openresty]#ll bin | grep openresty
lrwxrwxrwx.  1  root  root ...  openresty ->/usr/local/openresty/nginx/sbin/nginx
```

由于 OpenResty 整合了 Nginx，因此可以直接按照 Nginx 的方式来使用。下面为 OpenResty 整合的 Nginx 可执行文件添加到环境变量目录中，操作如下。

```
[root@localhost ~]#cd /usr/local/openresty/nginx/sbin
[root@localhost sbin]#ln -s `pwd`/nginx /usr/local/sbin/nginx
```

在安装 Nginx 时编写的/etc/init.d/nginx 服务脚本也可以直接改为 OpenResty 中的 Nginx 来使用，只需要修改其中的路径即可，如下所示。

```
DAEMON=/usr/local/openresty/nginx/sbin/nginx
```

如果是在全新的系统中安装 OpenResty，还需要在防火墙中开放 80 端口。

```
[root@localhost ~]#iptables -I INPUT -p tcp --dport 80 -j ACCEPT
[root@localhost ~]#service iptables save
```

完成上述操作后，就可以使用如下命令实现 OpenResty 的启动、重新加载配置和停止。

```
[root@localhost ~]#nginx
[root@localhost ~]#nginx -s reload
[root@localhost ~]#nginx -s stop
```

4. 访问测试

在浏览器中访问 Linux 服务器，测试 OpenResty 是否已经可以使用。如果出现如图 5-30 所示的页面，说明 OpenResty 当前正在运行。OpenResty 默认的站点目录为/usr/local/openresty/nginx/html，读者可以在该目录中存放一些文件进行访问测试。

图 5-30　OpenResty 访问测试

5.4.2　OpenResty 开发入门

OpenResty 主要应用在一些对性能要求非常高的场合，例如 WAF(Web Application Firewall，Web 应用防火墙)、CDN(Content Delivery Network，内容分发网络)调度、广告系统、消息推送系统、RESTful API(一种服务器接口技术)等。一些知名的互联网公司如 CloudFlare、奇虎 360、新浪、京东等都在使用 OpenResty 相关的技术。

OpenResty 平台离不开 Nginx 和 Lua 语言，读者可通过 GitHub 开源平台中的 openresty-best-practices(OpenResty 最佳实践)项目进行深入学习，如图 5-31 所示，本节仅介绍一些简单的应用。

1. 使用命令行程序

OpenResty 提供了命令行程序 bin/resty 用于直接执行 Lua 脚本或代码，具体示例如下。

```
[root@localhost ~]#cd /usr/local/openresty/bin
[root@localhost bin]#./resty -e 'print("hello")'
Hello
```

```
[root@localhost bin]#echo 'print("Hello")' >~/hello.lua
[root@localhost bin]#./resty ~/hello.lua
Hello
```

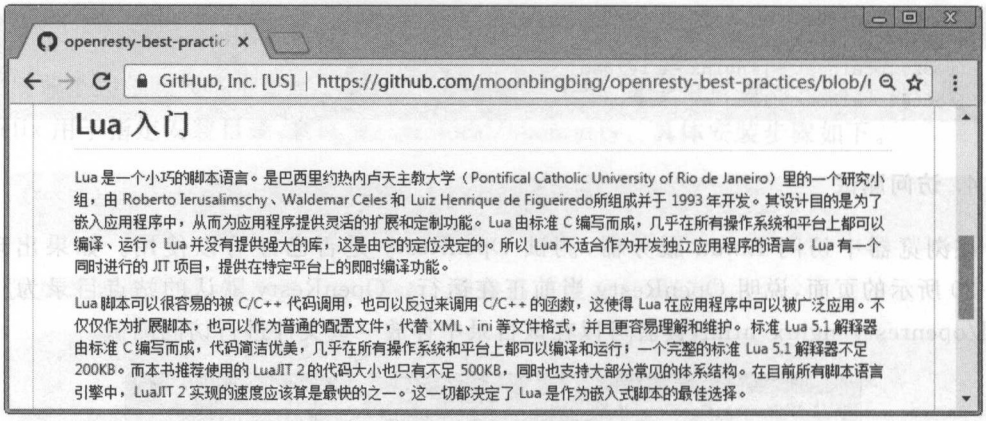


图 5-31 OpenResty 最佳实践

从上述示例可以看出，当指定选项-e 时，可以执行 Lua 代码 print("hello")，该代码用于输出一个字符串。也可以将 Lua 代码保存到一个文件中，通过 resty 程序执行。

OpenResty 还提供了一个查看帮助文档的工具 restydoc，通过选项-s 指定一个指令名称即可查看该指令的帮助文档。具体示例如下。

```
[root@localhost bin]#./restydoc -s proxy_pass
```

执行上述命令后的结果如图 5-32 所示。

```
[root@localhost bin]# ./restydoc -s proxy_pass
ngx_http_proxy_module(7)          nginx          ngx_http_proxy_module(7)

   proxy_pass
     syntax: proxy_pass "URL"

     context: location

     context: if in location

     context: limit_except

     Sets the protocol and address of a proxied server and an optional URI
     to which a location should be mapped.  As a protocol, ""http"" or
     ""https"" can be specified.  The address can be specified as a domain
     name or IP address, and an optional port:

         proxy_pass http://localhost:8000/uri/;
```

图 5-32 查看文档

restydoc 程序要求终端使用 UTF-8 编码(Xshell 默认为此编码)，并且系统中已经安装了新版本的 groff(GNU troff，一种文档排版程序)，若版本较旧时一些特殊字符将无法正常显示，需要下载最新版本编译安装，网址为 https://www.gnu.org/software/groff/。

2. 在配置文件中使用 Lua

OpenResty 支持直接在 Nginx 配置文件中编写 Lua 代码，或者指定某个 Lua 脚本文

件。在开始编写代码前，为了避免影响 Nginx 原有的文件，可以将工作目录临时更改为当前用户家目录。具体操作如下。

```
[root@localhost ~]#mkdir work
[root@localhost ~]#cd work
[root@localhost work]#mkdir logs conf
```

上述操作在家目录中创建了 work 目录，该目录将作为 Nginx 的工作目录。其中，work 目录下的 logs 用于保存相关日志，conf 目录用于保存配置文件。

接下来执行 vi conf/nginx.conf 创建配置文件，编写具体配置如下。

```
1   worker_processes  1;
2   error_log logs/error.log;
3   events {
4       worker_connections 1024;
5   }
6   http {
7       server {
8           listen 8080;
9           location / {
10              default_type text/html;
11              content_by_lua_block {
12                  ngx.say('<p>Hello World</p>')
13              }
14          }
15      }
16  }
```

在上述配置中，第 1～5 行是基本的配置；第 7～15 行部署了一个端口为 8080 的主机；第 10 行用于设置响应的内容格式；第 11～13 行通过 content_by_lua_block 指令实现在 Nginx 配置文件中编写 Lua 代码；第 12 行实现了输出一段 HTML 代码。

在完成配置文件的编写后，将 Nginx 工作目录变更为家目录中的 work 目录。如果当前 Nginx 服务已经启动，需要先执行 nginx -s stop 停止服务，然后执行如下操作启动 Nginx。

```
[root@localhost work]#nginx -p .
```

上述命令中，选项-p 用于改变 Nginx 工作目录，参数“.”表示改变为当前目录。命令执行后 Nginx 服务将会启动，通过 curl 命令可以访问进行测试，运行结果如下。

```
[root@localhost work]#curl http://localhost:8080
<p>Hello World</p>
```

接下来实现通过外部文件来执行 Lua 脚本。打开 conf/nginx.conf 文件，修改 server 配置如下。

```
1   server {
2       listen 8080;
```

```
    location /test.lua {
        default_type text/html;
        lua_code_cache off;
        content_by_lua_file test.lua;
    }
}
```

在上述配置中，content_by_lua_file 指令用于引入外部文件，lua_code_cache 用于开启或关闭 lua 文件缓存。在默认情况下会对 lua 文件进行缓存以提高效率，而在开发环境可以临时关闭缓存以方便测试。

执行 vi ~/work/test.lua 创建 Lua 脚本文件，编写代码如下。

```
ngx.say('<h1>Hello World</h1>')
```

更改 Nginx 配置文件后，通过如下命令重新载入配置文件。

```
[root@localhost work]#nginx -p . -s reload
```

通过 curl 命令测试新的配置是否生效，运行结果如下。

```
[root@localhost work]#curl http://localhost:8080/test.lua
<h1>Hello World</h1>
```

从运行结果可以看出，OpenResty 成功执行了 curl 命令请求的 lua 脚本文件。

本章小结

本章首先讲解 Nginx+PHP 环境的搭建，通过 FastCGI 实现整合。然后讲解 Nginx 与 Apache、Tomcat 的搭配，实现动静分离的环境。最后讲解 OpenResty 环境，在该环境下利用 Nginx+Lua 可以开发高性能的 Web 应用。通过本章的学习，读者可以掌握 Nginx 在 Web 服务器领域的主要应用。

课后练习

一、填空题

1. 在 LAMP 环境中，A 表示的软件是_____，用于提供 Web 访问。
2. 在 PHP 脚本中，若要查看 PHP 环境信息，可通过_____函数实现。
3. OpenResty 平台整合了_____和 LuaJit。

二、判断题

1. OpenResty 的 bin 目录下的 resty 程序可以执行 Lua 代码。（ ）
2. Apache 的分布式配置文件的文件名为. user. ini。（ ）
3. OpenResty 的 bin 目录下的 restydoc 用于查看文档。（ ）

4. 将 PHP 与 Apache 整合时，可以使用 apxs 编译 PHP 模块。（　　）

三、选择题

1. 在 FastCGI 常用环境变量中，保存客户端 IP 地址的是(　　)。

A. REQUEST_URI　　B. REMOTE_ADDR

C. REMOTE_PORT　　D. REQUEST_METHOD

2. 在 Tomcat 安装后，默认的站点目录为(　　)。

A. html　　B. htdocs　　C. webapps　　D. work

四、操作题

在搭建 Nginx＋PHP 环境时，Nginx 与 PHP 之间的 FastCGI 连接除了用本地 IP (127.0.0.1)的方式，还可以使用 Socket 方式。Unix Domain Socket 是一种进程间通信机制，在进行本地的两个进程之间通信时，相比使用本地 IP 的方式更有效率。请尝试在 Nginx＋PHP 环境中实现 Socket 方式通信。

第6章 负载均衡与缓存

学习目标

- 掌握反向代理与负载均衡的原理及配置；
- 掌握缓存的不同实现方式；
- 了解邮件服务配置相关指令。

对于一个大型网站来说，随着网站访问量的快速增长，单台服务器已经无法承担大量用户的并发访问，必须采用多台服务器协同工作，以提高计算机系统的处理能力、计算速度，从而满足当前业务量的需求。

那么如何实现服务器之间的协同功能呢？通过 Nginx 提供的反向代理和负载均衡功能，可以合理地完成业务的分配，提高网站的处理能力；同时利用缓存功能，还可以将不需要实时更新的动态页面输出结果，转化为静态网页形成缓存，从而提高网站的响应速度。本章将以实际应用为目标，从原理到具体实现深度剖析 Nginx 相关功能的配置。

6.1 反向代理

代理服务技术在 Nginx 服务器中占据着举足轻重的地位，这是由于 Nginx 中的许多应用功能都是从代理服务中衍生出来的，如负载均衡。因此，在学习如何配置负载均衡应用前，本节将对代理、反向代理的概念、区别、运行原理以及配置进行详细讲解。

6.1.1 代理与反向代理

代理也被称为正向代理，是一个位于客户端和目标服务器之间的代理服务器，客户端将发送的请求和指定的目标服务器提交给代理服务器，然后代理服务器向目标服务器发起请求，并将获得的响应结果返回给客户端的过程，具体如图 6-1 所示。

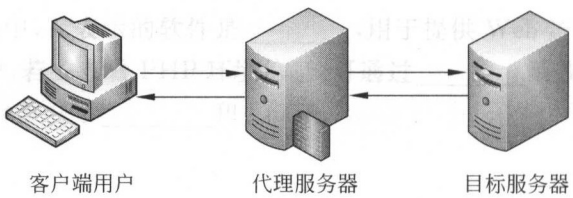


图 6-1 代理示意图

从图 6-1 可知，若一个用户 A 没有访问目标服务器的权限，但是该用户可以访问代理服

务器，且代理服务器又可以访问目标服务器。此时，就可以通过代理服务完成相应的请求，使用户获取到目标服务器响应的内容。需要注意的是，要想实现代理服务，客户端必须要进行一些特别的设置，由于这不是本节的重点，读者了解其运行原理即可。

相对于代理服务，反向代理对于客户端而言就是目标服务器，客户端向反向代理服务器发送请求后，反向代理服务器将该请求转发给内部网络上的后端服务器，并将从后端服务器上得到的响应结果返回给客户端。具体过程如图 6-2 所示。

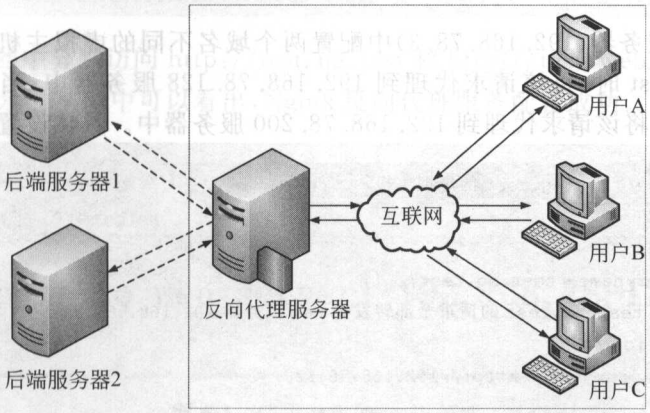


图 6-2　反向代理示意图

从图 6-2 可知，若用户 A、用户 B、用户 C 同时对反向代理服务器发送请求，反向代理服务器则根据其内部的具体配置，将用户的请求分发给后端服务器进行处理，并将后端服务器处理后的响应结果作为自己的响应结果返回给用户。反向代理服务器的整个处理过程，用户并不知情。因此，从上述对代理和反向代理的介绍可以总结出两者的特性，主要有以下几点。

- 安全性。正向代理的客户端能够在隐藏自身信息的同时访问任意网站，这给网络安全带来了极大的威胁。因此，在使用时必须采取安全措施以确保仅为经过授权的客户端用户提供服务。而反向代理的客户端用户只能通过外网来访问代理服务器，并且用户并不知道自己访问的是一个代理服务器，好处就是反向代理将真正的处理放在内网中，有效地提高了网络安全性。
- 功能性。正向代理的主要用途是为在防火墙内的局域网用户提供访问 Internet 的途径。而反向代理的主要用途是将防火墙后的服务器提供给 Internet 用户访问，还可以为多个后端服务器提供负载均衡功能、缓存功能等。

6.1.2　反向代理服务配置

在 Nginx 服务器中，反向代理的配置非常简单，最主要的指令就是 proxy_pass，用于设置后端服务器的地址。该地址中包括传输数据使用的协议、服务器主机名以及可选的 URI 资源等。在 Nginx 配置文件中，proxy_pass 指令通常在 location 块中进行设置。

下面通过一个简单的案例演示反向代理的配置，具体步骤如下。

1. 准备服务器

准备3台虚拟机，并全部安装Nginx服务器。其中，IP为192.168.78.3的服务器用作反向代理服务器，另外两台用作后端Web服务器，分别为192.168.78.128和192.168.78.200。

2. 配置反向代理

在反向代理服务器(192.168.78.3)中配置两个域名不同的虚拟主机，用于当用户请求http://test.ng.test时，将该请求代理到192.168.78.128服务器中；当用户请求http://test.web.com时，将该请求代理到192.168.78.200服务器中。具体设置如下。

```
1   #配置域名为test.ng.test的虚拟主机
2   server {
3       listen       80;
4       server_name  test.ng.test;
5       #域名test.ng.test的请求全部转发到Web服务器192.168.78.128
6       location / {
7           proxy_pass http://192.168.78.128;
8       }
9   }
10  #配置域名为test.web.com的虚拟主机
11  server {
12      listen       80;
13      server_name  test.web.com;
14      #域名test.web.com的请求全部转发到Web服务器192.168.78.200
15      location / {
16          proxy_pass http://192.168.78.200;
17      }
18  }
```

上述第2～9行和第11～18行分别用于配置域名为test.ng.test和test.web.com的虚拟主机。其中，第7行和第16行用于配置反向代理，proxy_pass指令后的http://表示传输数据所使用的协议，在服务器支持HTTPS服务的情况下，还可以将其指定为https://安全网络传输协议。关于Nginx的HTTPS服务的配置将会在后面的章节中讲解。

接着，在物理机中以管理员身份打开文本编辑器，编辑C:\Windows\System32\drivers\etc目录下的hosts文件，实现网站的域名访问。具体如下：

```
192.168.78.3 test.web.com
192.168.78.3 test.ng.test
```

完成上述配置后，当用户请求http://test.ng.test或http://test.web.com时，就会被反向代理服务器转发到后端的Web服务器192.168.78.128或192.168.78.200中进行处理。

3. 验证测试

在Web服务器192.168.78.128和192.168.78.200的网站根目录下定义默认访问文

件 index.html，用于区分不同的 Web 服务器。

(1) 在 Web 服务器 192.168.78.128 中编写 index.html，内容如下：

```
<h1>Welcome to web server1:192.168.78.128!</h1>
```

(2) 在 Web 服务器 192.168.78.128 中编写 index.html，内容如下：

```
<h1>Welcome to web server2:192.168.78.200!</h1>
```

最后，在浏览器中分别访问 http://test.ng.test 和 http://test.web.com 进行测试，如图 6-3 和图 6-4 所示。从图中可以看出，Nginx 反向代理服务配置成功。

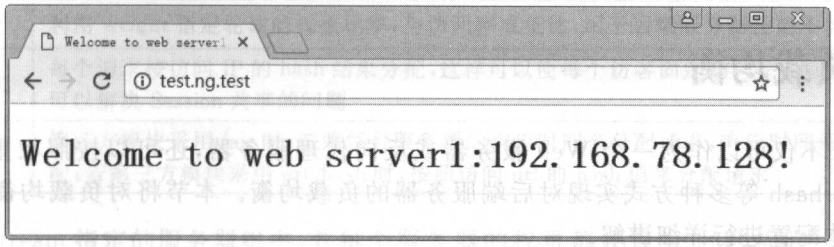


图 6-3　使用 web_server1 处理

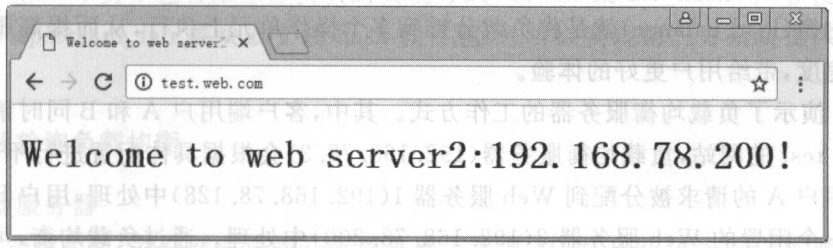


图 6-4　使用 web_server2 处理

除了上述使用的 proxy_pass 指令外，Nginx 中对于反向代理的设置还提供了很多辅助指令，其中常用的指令如表 6-1 所示。

表 6-1　其他常用反向代理指令

指　令	说　明
proxy_set_header	在将客户端请求发送给后端服务器之前，更改来自客户端的请求头信息
proxy_connect_timeout	配置 Nginx 与后端代理服务器尝试建立连接的超时时间
proxy_read_timeout	配置 Nginx 向后端服务器组发出 read 请求后，等待响应的超时时间
proxy_send_timeout	配置 Nginx 向后端服务器组发出 write 请求后，等待响应的超时时间
proxy_redirect	用于修改后端服务器返回的响应头中的 Location 和 Refresh

下面演示 proxy_set_header 指令的使用，如下配置实现了将客户端 IP 传递给后端服务器。

```
location / {
    proxy_pass http://192.168.78.200;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

在上述配置中，proxy_set_header 指令后的第 1 个参数用于表示字段名称，第 2 个参数表示字段值。其中，内置变量 $remote_addr 用于获取客户端真实的 IP 地址，$proxy_add_x_forwarded_for 用于在客户端请求头字段后添加客户端地址，使用逗号分隔，且当不存在客户端请求头字段时，该变量等同于变量 $remote_addr。

6.2 负载均衡

Nginx 不仅可以作为一个 Web 服务器或反向代理服务器，还可以按照权重、轮询、ip hash、URL hash 等多种方式实现对后端服务器的负载均衡。本节将对负载均衡实现的原理以及具体配置进行详细讲解。

6.2.1 什么是负载均衡

负载均衡(load balance)就是将负载分摊到多个操作单元上执行，从而提高服务的可用性和响应速度，带给用户更好的体验。

图 6-5 演示了负载均衡服务器的工作方式。其中，客户端用户 A 和 B 同时请求了域名为 test.ng.test 的网站，负载均衡服务器(192.168.78.3)会根据具体配置进行不同的分配。这里假设用户 A 的请求被分配到 Web 服务器 1(192.168.78.128)中处理，用户 B 的请求被分配到另一个闲置的 Web 服务器 2(192.168.78.200)中处理。通过负载均衡，可以将一台服务器的工作扩展到多台服务器中执行，提高整个网站的负载能力。

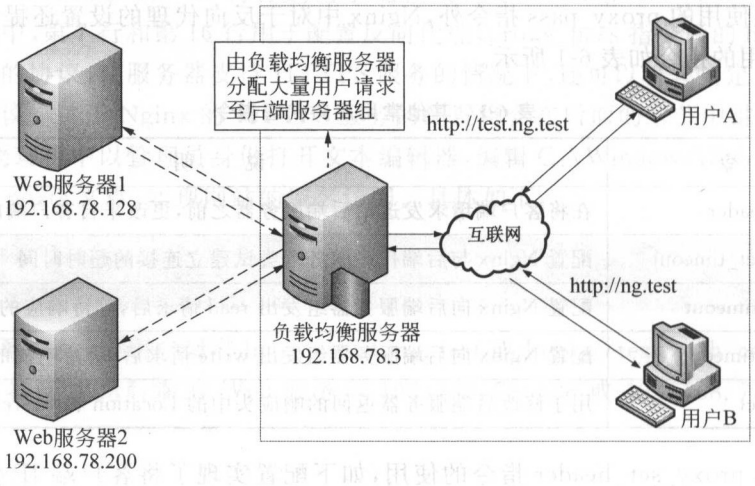


图 6-5 负载均衡示意图

6.2.2　负载均衡的配置

通过 Nginx 中的 upstream 指令可以实现负载均衡，在该指令中能够配置负载服务器组。目前负载均衡有 4 种典型的配置方式，分别为轮询方式、权重方式、ip_hash 方式，以及利用第三方模块的方式。关于每种配置方式的特点，详见表 6-2。

表 6-2　不同负载均衡配置特点

配置方式	说　明
轮询方式	负载均衡默认设置方式，每个请求按照时间顺序逐一分配到不同的后端服务器进行处理，如果有服务器宕机，会自动剔除
权重方式	利用 weight 指定轮询的权重比率，与访问率成正比，用于后端服务器性能不均的情况
ip_hash 方式	每个请求按访问 IP 的 hash 结果分配，这样可以使每个访客固定访问一个后端服务器，可以解决 Session 共享的问题
第三方模块	第三方模块采用 fair 时，按照每台服务器的响应时间来分配请求，响应时间短的优先分配；若第三方模块采用 url_hash 时，按照访问 url 的 hash 值来分配请求

在 upstream 指定的服务器组中，若每个服务器的权重都设置为 1(默认值)时，表示当前的负载均衡是一般轮询方式。

另外，Nginx 本身不包含第三方模块的实现方式，如 fair 或 url_hash 等，在使用时必须下载对应的 upstream_fair 模块或安装 hash 软件包，才可以实现第三方模块提供的负载均衡配置。

1. 一般轮询负载均衡

1) 准备服务器

准备 3 台虚拟机，并全部安装 Nginx 服务器。其中，IP 为 192.168.78.3 的服务器用作负载均衡服务器，另外两台用作后端 Web 服务器，IP 分别为 192.168.78.128 和 192.168.78.200。

2) 配置一般轮询负载均衡

```
1   #配置域名为 test.ng.test 的虚拟主机
2   server {
3       listen       80;
4       server_name  test.ng.test;
5       location / {
6           proxy_pass http://web_server;
7       }
8   }
9   #配置负载均衡服务器组
10  upstream web_server {
11      server 192.168.78.128;
12      server 192.168.78.200;
13  }
```

在上述配置中，第 6 行用于指定代理的 URL，第 10～13 行用于设置负载均衡服务器

组。其中，upstream 指令后的 web_server 表示代理的服务器主机名，用于第 6 行的 proxy_pass 指令执行反向代理时使用；第 11～ 12 行利用 server 指令在 upstream 块中配置了后端 Web 服务器，这些服务器可以有一个或多个，从而形成负载均衡服务器组。

3）验证测试

在 Web 服务器 192.168.78.128 和 192.168.78.200 的网站根目录下定义默认访问文件 index.html，用于区分当前处理用户请求的是哪一台 Web 服务器。

（1）在 Web 服务器 192.168.78.128 中编写 index.html，内容如下：

```
<h1>web server:192.168.78.128!</h1>
```

（2）在 Web 服务器 192.168.78.200 中编写 index.html，内容如下：

```
<h1>web server:192.168.78.200!</h1>
```

在浏览器中访问 http://test.ng.test，第一次访问的结果如图 6-6 所示；刷新一次后，访问结果如图 6-7 所示。接着尝试不断刷新，如果测试结果在图 6-6 和图 6-7 之间交替，说明负载均衡服务器根据每个请求按照时间顺序逐一分配到 192.168.78.128 和 192.168.78.200 两台 Web 服务器中执行。

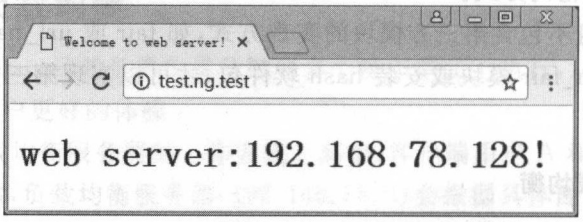


图 6-6 第 1 次访问结果

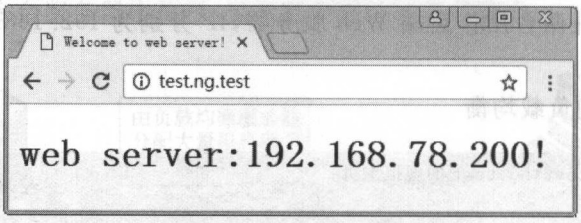


图 6-7 第 2 次访问结果

值得一提的是，如果 Nginx 检测到后端某台服务器宕机，则会在负载均衡时自动剔除该服务器。下面关闭 Web 服务器 192.168.78.200 进行测试，通过反复刷新 http://test.ng.test 观察浏览器显示结果。如果网页中一直显示当前后端服务器为 192.168.78.128，说明 192.168.78.200 服务器已经自动被剔除。

2. 加权轮询负载均衡

如果负载均衡服务器组中的服务器硬件配置强弱不一，则可以通过 weight 参数设置权重大小。对于配置较好的服务器，可以为其设置成比较高的权值，对于配置较差的服务器，

可以为其分配较小的权值。通过加权轮询，可以让每台服务器承担与之硬件配置相符的工作量，从而在整体上发挥最佳的效果。

接下来，为上一小节中配置的服务器组设置权重，实现加权轮询，具体如下。

```
upstream web_server {
    server 192.168.78.128 weight=1;
    server 192.168.78.200 weight=3;
}
```

在上述配置中，weigth 参数表示权值，权值越高则被分配到的概率越大。在负载均衡的过程中，Nginx 将按照平滑加权轮询算法进行具体分配。其中，权值总和为一个循环，这里的配置就是以 4 次请求为一个循环，在循环过程中，服务器 192.168.78.128 会在 4 次请求中被分别到 1 次，服务器 192.168.78.200 则会被分配到 3 次，但是 3 次被选取的机会并不会连续执行，而是按照算法分散执行。

按照上述配置完成后，通过浏览器访问 4 次 http://test.ng.test，可以看到其中一次显示如图 6-6 所示，其余 3 次显示如图 6-7 所示。

除此之外，还可以设定每台 Web 服务器在负载均衡调度中的状态，常用的参数说明如表 6-3 所示。

表 6-3　常用状态参数

配置方式	说　明
max_fails	允许请求失败的次数，默认为 1。当超过最大次数时，返回 proxy_next_upstream 指令定义的错误
fail_timeout	在经历了 max_fails 次失败后，暂停服务的时间。且在实际应用中 max_fails 一般与 fail_timeout 一起使用
backup	预留的备份机器
down	表示当前的 server 暂时不参与负载均衡

在表 6-3 中，设置为 backup 的服务器，只有当其他所有的非 backup 机器出现故障或者忙碌的情况下，才会请求 backup 服务器，因此这台服务器的压力最小。

下面再准备一台虚拟机，用作预留的备份机器，然后将负载均衡服务器组，修改成如下形式：

```
upstream web_server {
    server 192.168.78.128 weight=1 max_fails=1 fail_timeout=2;
    server 192.168.78.200 weight=3 max_fails=2 fail_timeout=2;
    server 192.168.78.201 backup;
}
```

在上述配置中，第 2～3 行设置了允许请求的最大失败次数和请求失败后暂停服务的时间；第 4 行设置了一台备份机器，用于在前面两台 Web 服务器出现故障的情况下，为用户继续提供服务。

为了验证和测试，在备份服务器(192.168.78.201)的网站中添加 index.html 页面，具体如下。

```
<h1>backup server:192.168.78.201!</h1>
```

在关闭后端服务器 192.168.78.128 和 192.168.78.200 后，访问网站 http://test.ng.test，可以看到如图 6-8 所示的结果，说明当前备份服务器已经工作。若在配置中去掉备份服务器，再次访问则会出现 502 Bad Gateway 错误。因此，在配置负载均衡服务时，利用备份服务器可以应对一些意外情况，提高服务的可用性。

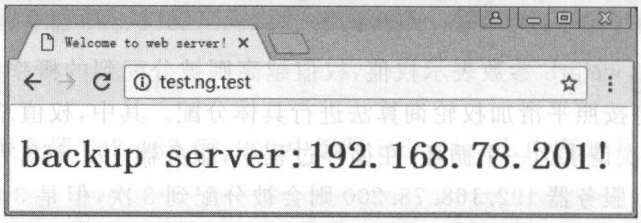


图 6-8 负载均衡备份服务器

3. ip_hash 负载均衡

ip_hash 方式的负载均衡，是将每个请求按照访问 IP 的 hash 结果分配，这样就可以使来自同一个 IP 的客户端用户固定访问一台 Web 服务器，有效地解决了动态网页存在的 Session 共享问题。下面将上述负载均衡服务器组的设置修改成如下形式：

```
upstream web_server {
    ip_hash;
    server 192.168.78.128;
    server 192.168.78.200;
    server 192.168.78.3 down;
}
```

在上述配置中，upstream 块中的 ip_hash 指令用于标识当前负载均衡的处理方式。其中，对于一个暂时性宕机的服务器，可以使用 down 参数标识出来，这样在负载均衡时，就会忽略该服务器的分配。

需要注意的是，在使用 ip_hash 方式处理负载均衡时，Web 服务器在负载均衡列表中的状态不能使用 weight 和 backup 设置。

接着，使用浏览器访问 http://test.ng.test 进行测试，同时要保证当前 Web 服务器全部正常开启。经过多次刷新后，可发现处理该用户的服务器一直是 192.168.78.128。为了进一步验证，可以在服务器 192.168.78.128 中使用以下命令查看连接数。

```
[root@localhost ~]#netstat -n | grep :80 | wc -l
```

在上述命令中，选项 n 用于显示 IP 地址和端口号，grep :80 表示只显示 80 端口的服务，wc -l 用于统计查询的结果。执行上述命令后，参考结果如图 6-9 所示。

```
[root@localhost ~]# netstat -n | grep :80 | wc -l
14
[root@localhost ~]#
```

图 6-9 统计访问连接数

值得一提的是，由于 ip_hash 方式为每一个用户 IP 绑定一个 Web 服务器处理，将会导致某些 Web 服务器接收的请求多，某些 Web 服务器接到的请求少，无法保证 Web 服务器的负载均衡。因此，建议只在必要的情况下使用这种方式。

4. 利用第三方模块

第三方提供的方式有多种，下面以 fair 方式为例，按照 Web 服务器的响应时间实现负载均衡，响应时间短的优先分配。具体实现步骤如下。

1）备份已安装的 Nginx

由于使用第三方模块需要重新编译 Nginx，所以在此之前，需要关闭已经开启的 Nginx 进程，对已经安装好的 Nginx 进行备份，便于恢复。

```
[root@localhost ~]#cp -r /usr/local/nginx /usr/local/nginx_old
```

2）重新编译安装 Nginx

在开源软件平台 Github 中可以获取 fair 模块。下载 nginx-upstream-fair-master. zip 模块文件到 root 目录下，然后将其解压并重命名为 nginx-upstream-fair，具体命令如下。

```
[root@localhost ~]#unzip nginx-upstream-fair-master.zip
[root@localhost ~]#mv nginx-upstream-fair-master nginx-upstream-fair
```

值得一提的是，在使用 unzip 解压文件时，若提示没有该命令，则需要使用 yum 安装后再执行，执行完成后的效果如图 6-10 所示。

```
[root@localhost ~]# unzip nginx-upstream-fair-master.zip
Archive:  nginx-upstream-fair-master.zip
a18b4099fbd45811983200e098b6f0c8efed4bc
   creating: nginx-upstream-fair-master/
  inflating: nginx-upstream-fair-master/.gdbinit
  inflating: nginx-upstream-fair-master/README
  inflating: nginx-upstream-fair-master/config
  inflating: nginx-upstream-fair-master/ngx_http_upstream_fair_module.c
```

图 6-10　解压文件

接着，进入 Nginx 文件的解压目录，在编译选项中添加对 nginx-upstream-fair 模块的支持，具体命令如下。

```
[root@localhost ~]#cd nginx-1.10.1
[root@localhost nginx-1.10.1]#./configure \
--prefix=/usr/local/nginx \
--with-http_ssl_module \
--add-module=/root/nginx-upstream-fair
[root@localhost nginx-1.10.1]#make && make install
```

上述命令执行后，即可完成 Nginx 和第三方模块 fair 的编译和安装。

3）配置 fair 方式的负载均衡

打开新安装的 Nginx 配置文件，在 http 块下实现 fair 方式的负载均衡。具体配置如下。

```
#配置域名为test.ng.test的虚拟主机
server {
    listen       80;
    server_name  test.ng.test;
    location / {
        proxy_pass http://web_server;
    }
}
#配置fair方式的负载均衡
upstream web_server {
    server 192.168.78.128;
    server 192.168.78.200;
    fair;
}
```

在上述配置中，若要负载均衡采用fair方式执行，只需在upstream服务器列表中添加fair指令即可。

4）验证测试

为了验证fair模块是否能够根据后端服务器的响应时间负载均衡，可以通过PHP延长服务器的响应时间。在Web服务器192.168.78.128中配置PHP后，按照如下步骤进行操作。

(1) 在Web服务器192.168.78.128中，编写index.php文件，内容如下。

```
<?php sleep(10); ?>
<h1>web server:192.168.78.128!</h1>
```

上述代码中，sleep(10)用于使服务器延迟10s后响应。

(2) 在Web服务器192.168.78.200中，编写index.php文件，内容如下。

```
<h1>web server:192.168.78.200!</h1>
```

接下来访问域名为http://test.ng.test的网站。通过反复测试可以看出，按上述配置两次请求为一个循环，在每次循环过程中，响应快的服务器192.168.78.200都会优先分配，接着才会分配到响应速度较慢的服务器192.168.78.128。这里为了方便测试仅设置了两组负载服务器，读者可测试多个负载服务器组，即可以看出此负载均衡方式的优点。

6.3 缓存配置

对于一个含有大量内容的网站来说，随着访问量的增多，对于经常被用户访问的内容，若每一次都要到后端服务器中获取，会给服务器造成很大的压力。为此，利用反向代理服务器对访问频率较多的内容进行缓存，有利于节省后端服务器的资源。Nginx提供了两种Web缓存方式，一种是永久性缓存，另一种是临时性缓存。

6.3.1　缓存实现原理

Web 缓存服务器位于内容源 Web 服务器和客户端之间，当客户端用户访问一个 URL 时，Web 缓存服务器就会请求相应的内容源 Web 服务器，并将响应的信息缓存至内存或磁盘；然后，当下一个请求到来时，如果访问的是相同的 URL，Web 缓存服务器会直接将已缓存的内容输出给客户端，而不用再次向内容源 Web 服务器发送请求。

利用缓存服务器，可以有效降低内容源服务器和数据库的负载，提高用户访问的响应速度，参见图 6-11。

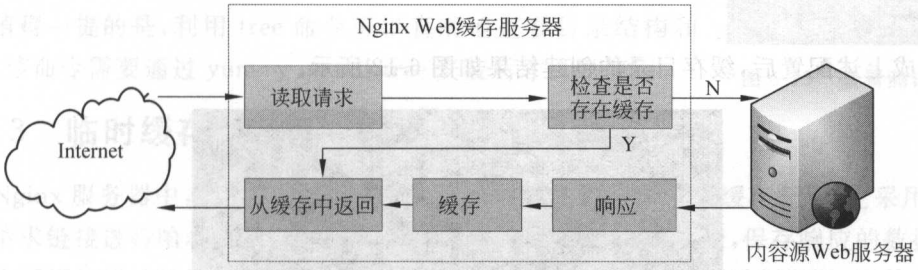


图 6-11　缓存示意图

在图 6-11 中，当用户向 Web 缓存服务器中发送请求时，缓存服务器要检查一下当前 URL 请示是否已经存在缓存，若存在则直接返回给用户。否则，向内容源 Web 服务器发送请求，获取响应结果，将其缓存并返回给发送请求的客户端。

6.3.2　永久缓存配置

Nginx 提供的 proxy_store 指令可以用于将内容源服务器响应的内容缓存到本地，若不手动删除，该缓存文件会一直生效。因此，永久缓存方式适用于缓存网站中几乎不会更改的一些内容。

下面通过一个简单的案例演示永久缓存的配置，具体步骤如下。

1. 准备服务器

准备 2 台虚拟机，并安装 Nginx。其中一台作为 Web 缓存服务器(192.168.78.3)，另外一台用作内容源 Web 服务器(192.168.78.128)。

2. 缓存配置

打开 Web 缓存服务器(192.168.78.3)的配置文件 nginx.conf，添加如下配置。

```
server {
    listen       80;
    server_name  192.168.78.3;
    location / {
        root cache;
        proxy_store on;
```

```
7            proxy_store_access user:rw group:rw all:r;
8            proxy_temp_path cache_tmp;
9            proxy_pass http://192.168.78.128;
10       }
11   }
```

上述第 5 行配置，用于指定缓存文件的保存目录，这里将其设定在 Nginx 安装目录下的 cache 目录中，需要用户手动创建，创建后修改此目录的用户权限，要求与 Nginx 工作进程的用户相同(如 nobody)。第 6 行用于开启本地缓存，第 7 行设置缓存的读写规则，第 8 行设置反向代理时接收的数据临时存储文件的目录，该目录会由 Nginx 在配置生效后自动创建。

完成上述配置后，缓存目录的创建结果如图 6-12 所示。

```
[root@localhost nginx]# ll
total 44
drwxr-xr-x. 3 nobody root 4096 Oct 21 10:30 cache
drwx------. 2 nobody root 4096 Oct 21 10:30 cache_tmp
drwx------. 2 nobody root 4096 Oct 19 23:14 client_body_temp
drwxr-xr-x. 2 root   root 4096 Oct 19 19:56 conf
drwx------. 2 nobody root 4096 Oct 19 20:03 fastcgi_temp
drwxr-xr-x. 2 nobody root 4096 Oct 19 23:24 html
drwxr-xr-x. 2 root   root 4096 Oct 19 20:03 logs
drwx------. 2 nobody root 4096 Oct 19 23:27 proxy_temp
drwxr-xr-x. 2 root   root 4096 Oct 19 19:56 sbin
drwx------. 2 nobody root 4096 Oct 19 20:03 scgi_temp
drwx------. 2 nobody root 4096 Oct 19 20:03 uwsgi_temp
[root@localhost nginx]#
```

图 6-12　查看缓存目录

需要注意的是，上述配置虽然能够将文件缓存在本地，但是客户端每次请求时，Nginx 仍然会向后端服务器获取文件。为了避免这种情况的发生，需要先判断缓存文件是否存在，具体配置如下。

```
#利用正则表达式匹配缓存目录中的文件、目录或符号链接是否存在
if(!-e $request_filename){
    proxy_pass http://192.168.78.128;
}
```

上述配置中，!-e 表示检查一个文件、目录或符号链接是否存在，当不存在时就执行{}中的指令。其中，内置变量 $request_filename 表示当前请求的文件路径或 URI。例如，当客户端向 Web 缓存服务器发送 http://192.168.78.3/test/index.html 请求时，变量 $request_filename 的值为/test/index.html，如果缓存目录 cache 中没有用户请求的缓存文件，Nginx 会到后端服务器 http://192.168.78.128/test/index.html 中请求，然后将响应结果进行缓存。

3. 验证测试

在内容源 Web 服务器(192.168.78.128)中准备一些测试文件用于访问测试，具体步骤如下。

(1) 在站点目录下创建 index.html 文件，编写如下内容。

```
<h1>192.168.78.128/index.html!</h1>
```

(2) 在站点目录下创建 test 目录，在该目录下上传一张图片 apple.jpg，并编写如下内容的 test.html 文件。

```
<h1>192.168.78.128/test/test.html!</h1>
<img src="apple.jpg" />
```

接下来，通过浏览器分别访问 http://192.168.78.3/index.html 和 http://192.168.78.3/test/test.html，然后到 Web 缓存服务器(192.168.78.3)设置的缓存目录 cache 中查看缓存结果，如图 6-13 所示。

```
[root@localhost cache]# tree
.
├── index.html
└── test
    ├── apple.jpg
    └── test.html

1 directory, 3 files
[root@localhost cache]#
```

图 6-13　查看测试结果

值得一提的是，利用 tree 命令可以直观地查看目录结构和文件，该命令需要通过 yum -y install tree 安装后才可以使用。

6.3.3　临时缓存配置

Nginx 服务器中，还有一种使用 proxy_cache 指令设置的临时缓存配置，它采用 md5 算法将请求链接进行哈希(hash)后，根据具体配置生成缓存文件目录，保存响应的数据。

为了更加清楚地了解临时缓存配置，下面通过一个简单的案例进行演示，具体如下。

(1) 准备服务器

准备 2 台虚拟机，并全部安装 Nginx 服务器。其中一台作为 Web 缓存服务器(192.168.78.3)，另一台作为内容源 Web 服务器(192.168.78.128)。

(2) 缓存配置

打开 Web 缓存服务器(192.168.78.3)的配置文件 nginx.conf，在 http 块中添加如下配置。

```
1    #代理临时目录
2    proxy_temp_path  /usr/local/nginx/proxy_temp_dir;
3    #Web 缓存目录和参数设置
4    proxy_cache_path /usr/local/nginx/proxy_cache_dir levels=1:2 keys_zone=cache_one:50m
     inactive=1m max_size=500m;
```

上述第 2 行指令，用于设置缓存服务器(192.168.78.3)接收内容源服务器响应内容时使用的临时目录。第 4 行指令，用于设置缓存目录。其中，proxy_cache_path 指令相关参数的含义如下。

- /usr/local/nginx/proxy_cache_dir 参数：表示用户自定义的缓存文件保存目录。
- levels 参数：表示缓存目录下的层级目录结构，它是根据哈希后的请求 URL 地址创建的，目录名称从哈希后的字符串结尾处开始截取。

  假设哈希后的请求链接地址为 af7098a15e430326197ee01516fdace0，则 levels＝1:2 表示，第 1 层子目录的名称是长度为 1 的字符 0，第 2 层子目录的名称是长度为 2 的字符 ce。
- keys_zone 参数：指定缓存区名称及大小，例如，cache_one:50m 表示缓存区名称为 cache_one，在内存中的空间是 50MB。
- inactive 参数：表示主动清空在指定时间内未被访问的缓存。例如，1m 清空在 1 分

钟内未被访问过的缓存，1h表示1小时，1d表示1天等。

- max_size参数：表示指定磁盘空间大小。例如，500m、10g。

需要注意的是，Nginx在进行缓存时，首先会被写入proxy_temp_path指定的临时目录中，因此建议proxy_cache_path和proxy_temp_path指令设置的目录应在同一个文件系统中，避免不同文件系统之间的磁盘I/O消耗。

(3) 接着，在server块中添加临时缓存的相关配置，具体如下：

```
1   server {
2       listen       80;
3       server_name  test.ng.test;
4       #增加两个响应头信息,用于获知访问的服务器地址与缓存是否成功
5       add_header X-Via $server_addr;
6       add_header X-Cache $upstream_cache_status;
7       location / {
8             #设置缓存区域名称
9             proxy_cache cache_one;
10            #以域名、URI、参数组合成Web缓存的Key值,Nginx根据Key值哈希
11            proxy_cache_key $host$uri$is_args$args;
12            #对不同的HTTP状态码设置不同的缓存时间
13            proxy_cache_valid 200 10m;          #200缓存10分钟
14            proxy_cache_valid 304 1m;           #304缓存1分钟
15            proxy_cache_valid 301 302 1h;       #301、302缓存1小时
16            proxy_cache_valid any 1m;           #其他未设置的状态码缓存1分钟
17            #设置反向代理
18            proxy_pass      http://192.168.78.128;
19      }
20  }
```

上述第9行，用于设置缓存区域名称；第11行用于设置hash的Key值组成规则，在省略的情况下，Nginx将使用默认的Key值组成规则。第13～16行设置，通过proxy_cache_valid指令对不同的HTTP状态码设置不同的缓存时间，该指令的第1个参数表示状态码，第2个参数表示缓存时间。

关于proxy_cache_key指令参数中使用的具体内置变量的说明如下。

- $host：服务器的域名，如test.ng.test。
- $uri：域名和参数之间的部分，如/index.html。
- $is_args：有URL参数时，则值为?，否则为空字符串。
- $args：保存URL参数，如a=1&b=2，没有参数时为空字符串。
- 利用$is_args和$args，可以实现根据不同URL参数缓存不同文件。

为了便于在浏览器端查看是否正确缓存，第5～6行配置通过add_header指令添加了两个响应消息头。其中X-Via表示服务器地址，利用内置变量$server_addr获取，另一个X-Cache表示资源缓存状态，利用内置变量$upstream_cache_status获取。$upstream_cache_status的返回值有7个，如表6-4所示。

表 6-4　缓存状态返回值

返　回　值	说　　明
HIT	缓存命中
MISS	未命中，请求被传送到后端
EXPIRED	缓存已经过期，请求被传送到后端
UPDATING	正在更新缓存，将使用旧的应答
STALE	无法从后端服务器更新缓存时，返回了旧的缓存内容（可通过 proxy_cache_use_stale 指令配置）
BYPASS	缓存被绕过了（可通过 proxy_cache_bypass 指令配置）
REVALIDATED	启用 proxy_cache_revalidate 指令后，当缓存内容过期时，Nginx 通过一次 If-Modified-Since 的请求头去验证缓存内容是否过期，此时会返回该状态

需要注意的是，对于用户的请求，仅在处理成功的情况下，才会在浏览器的 Response Headers 中查看到 add_header 指令设置的响应消息头。

（4）访问测试。

设置完成后，平滑重启 Nginx，使配置生效。在内容源服务器（192.168.78.128）中放置几个测试文件，用于访问测试。接下来，通过浏览器端打开 http://test.ng.test/index.html 后，刷新一次网页，然后按 F12 键打开开发工具，选择 Network 选项，单击 index.html 找到 Response Headers 标签选项，查看 X-Via 和 X-Cache 的值，如果看到如图 6-14 所示的效果则表明设置缓存成功。

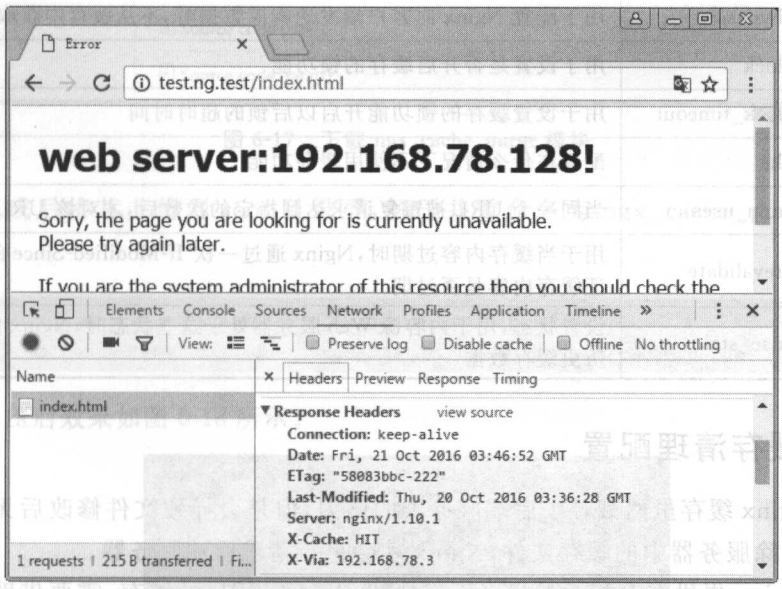


图 6-14　浏览器访问测试

接着，在 Web 缓存服务器（192.168.78.3）中，切换到 proxy_cache_dir 指令设置的缓存目录中查看当前缓存的文件内容，如图 6-15 所示。

从 tree 命令显示的树状结构图中可以看出，缓存目录的结构是按照 levels 的设置保存

的，第 1 层目录通过文件名的最后一个字符命名，第 2 层目录通过文件名的最后 2～3 个字符命名。

在缓存配置中，proxy_cache_path 的参数 inactive，设置了缓存文件在指定时间内未被访问将被自动删除，这里设置的是 1 分钟。缓存被自动删除后的效果如图 6-16 所示。

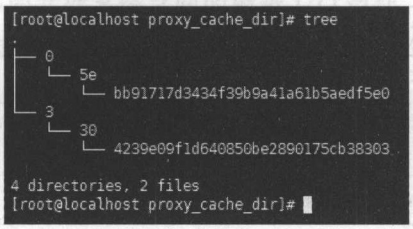


图 6-15 查看缓存目录一

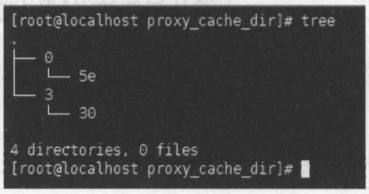


图 6-16 查看缓存目录二

为了进一步验证缓存自动删除的时间，接下来通过浏览器访问 http://test.ng.test/index.html，在打开网页后等待半分钟刷新一次，然后查看缓存目录中的文件。如果缓存文件仍然需要等待 1 分钟才会自动删除，则充分说明 inactive 参数设置的时间是从缓存文件最后一次被访问后才开始计算的。

除了上述使用的基本缓存配置指令外，Nginx 中还提供许多其他相关的指令，这里不再一一举例演示。为方便读者的学习，表 6-5 展示了一些常用的缓存配置指令。

表 6-5 常用缓存配置指令

指　令	说　明
proxy_cache_bypass	用于配置 Nginx 向客户端发送响应数据时，不从缓存中获取的条件
proxy_cache_lock	用于设置是否开启缓存的锁功能
proxy_cache_lock_timeout	用于设置缓存的锁功能开启以后锁的超时时间
proxy_no_cache	配置在什么情况下不使用缓存功能
proxy_cache_min_uses	当同一个 URL 被重复请求达到指定的次数后，才对该 URL 进行缓存
proxy_cache_revalidate	用于当缓存内容过期时，Nginx 通过一次 If-Modified-Since 的请求头去验证缓存内容是否过期
proxy_cache_use_stale	设置状态，用于内容源 Web 服务器处于这些状态时，Nginx 向客户端响应历史缓存数据

6.3.4 缓存清理配置

利用 Nginx 缓存虽然减轻了后端服务器的压力，但是会导致文件修改后无法及时更新缓存，只有删除服务器中的缓存文件，Nginx 才会重新请求后端服务器。

目前，Nginx 提供缓存相关的指令不支持清理指定 URL 的缓存，需要借助第三方模块才可以实现。本节以 ngx_cache_purge 模块为例进行讲解，下面演示具体的操作步骤。

1. 备份已安装的 Nginx

在添加 ngx_cache_purge 模块前，关闭 Nginx 服务，备份已有的 Nginx 目录。

```
[root@localhost ~]#cp -r  /usr/local/nginx /usr/local/nginx_old2
```

2. 重新编译安装 Nginx

在 GitHub 平台可以获取 ngx_cache_purge 模块的源代码，参考下载地址为 https://github.com/FRiCKLE/ngx_cache_purge/releases。这里以 2.3 版本的 zip 格式的文件为例讲解，如图 6-17 所示。

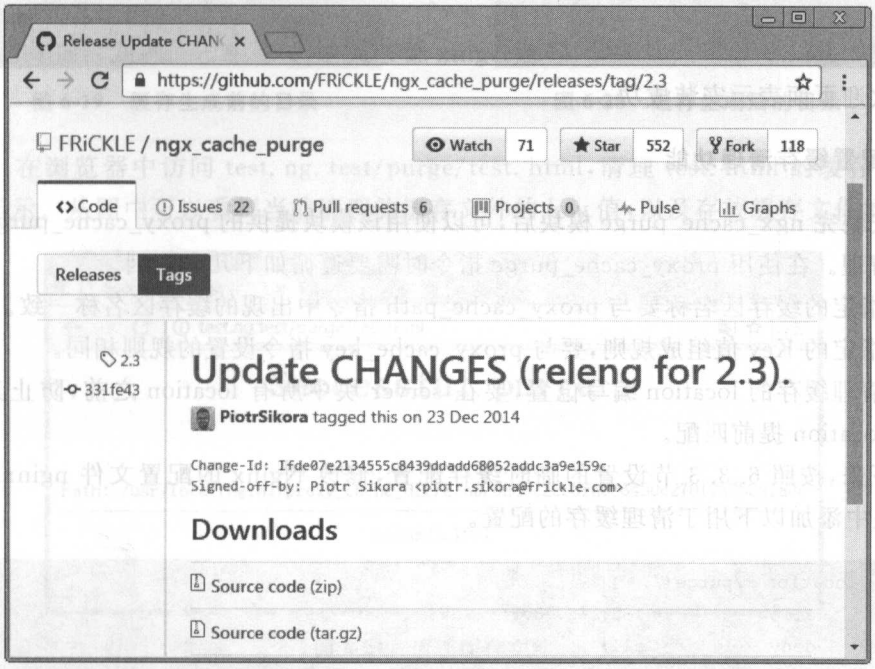


图 6-17　下载 ngx_cache_purge 模块

下载完成后将其上传到 root 目录下，解压并重命名为 ngx_cache_purge，具体命令如下。

```
[root@localhost ~]#unzip ngx_cache_purge-master.zip
[root@localhost ~]#mv ngx_cache_purge-master /usr/local/ngx_cache_purge
```

文件解压后效果如图 6-18 所示。

```
[root@localhost ~]# unzip ngx_cache_purge-master.zip
Archive:  ngx_cache_purge-master.zip
331fe43e8d9a3d1fa5e0c9fec7d3201d431a9177
   creating: ngx_cache_purge-master/
  inflating: ngx_cache_purge-master/CHANGES
  inflating: ngx_cache_purge-master/LICENSE
  inflating: ngx_cache_purge-master/README.md
  inflating: ngx_cache_purge-master/TODO.md
  inflating: ngx_cache_purge-master/config
  inflating: ngx_cache_purge-master/ngx_cache_purge_module.c
   creating: ngx_cache_purge-master/t/
  inflating: ngx_cache_purge-master/t/proxy1.t
  inflating: ngx_cache_purge-master/t/proxy1_vars.t
  inflating: ngx_cache_purge-master/t/proxy2.t
  inflating: ngx_cache_purge-master/t/proxy2_vars.t
```

图 6-18　解压文件

接着，进入 Nginx 文件的解压目录，在./configure 时添加对 nginx-cache_purge 模块的支持，完成 Nginx 编译选项的配置，具体命令如下。

```
[root@localhost ~]#cd nginx-1.10.1
[root@localhost nginx-1.10.1]#./configure \
--prefix=/usr/local/nginx \
--with-http_ssl_module \
--add-module=/usr/local/ngx_cache_purge
[root@localhost ~]#make && make install
```

完成 Nginx 的编译和安装后，启动 Nginx 服务，在浏览器中访问测试。如果看到 Nginx 的默认欢迎页面表示安装成功。

3. 配置缓存清理功能

在安装完 ngx_cache_purge 模块后，可以使用该模块提供的 proxy_cache_purge 指令实现缓存清理。在使用 proxy_cache_purge 指令时需要遵循如下几个规则。

- 指定的缓存区名称要与 proxy_cache_path 指令中出现的缓存区名称一致。
- 指定的 Key 值组成规则，要与 proxy_cache_key 指令设置的规则相同。
- 清理缓存的 location 编写位置，要在 server 块中所有 location 之前，防止其他正则 location 提前匹配。

接下来，按照 6.3.3 节设置的临时缓存配置，修改 Nginx 的配置文件 nginx.conf，在 server 块中添加以下用于清理缓存的配置。

```
1   location ~/purge(/.*){
2       allow           192.168.78.1;
3       deny            all;
4       proxy_cache_purge cache_one $host$1$is_args$args;
5   }
```

上述第 1 行，用于以正则方式匹配用户的请求，请求地址符合/purge/URI 形式时，就会清理 URI 对应的缓存文件。第 2～3 行用于指定允许清理缓存的客户端，此处表示仅允许 IP 地址为 192.168.78.1 的客户端清理缓存。第 4 行通过 proxy_cache_purge 指令，指定名称为 cache_one 的缓存区，和待清理缓存文件的 Key 值组成规则。其中，$1表示 location 正则表达式中的子模式“(/.*)”的匹配结果，对应生成缓存 Key 时的$uri。

4. 访问测试

在内容源服务器的网站根目录中，编写测试文件 test.html，具体内容如下。

```
<!DOCTYPE html>
<html>
    <head><title>Welcome to nginx!</title></head>
    <body><h1>Welcome to nginx!</h1></body>
</html>
```

为了测试 test.html 的缓存文件是否已经生成或被清理，应提前清空 proxy_cache_dir

目录，如图 6-19 所示。

然后，在浏览器中访问 test. ng. test/test. html 后，再次查看缓存目录，如图 6-20 所示。此时可以看到哈希后生成的缓存文件名称及目录结构。

```
[root@localhost proxy_cache_dir]# tree
.

0 directories, 0 files
[root@localhost proxy_cache_dir]#
```

图 6-19　缓存生成前的目录

```
[root@localhost proxy_cache_dir]# tree
.
├── 7
│   └── 79
│       └── c87c8bc45151b83f34296400d71c1797
└── e
    └── a8
        └── 0adb22cbbfa0e6430c27012a2539fa8e

4 directories, 2 files
[root@localhost proxy_cache_dir]#
```

图 6-20　缓存生成后的目录

最后，在浏览器中访问 test. ng. test/purge/test. html，清理 test. html 的缓存文件，如图 6-21 所示。从图中可以看到当前清理的缓存文件的 key 值，以及存放缓存文件的路径。

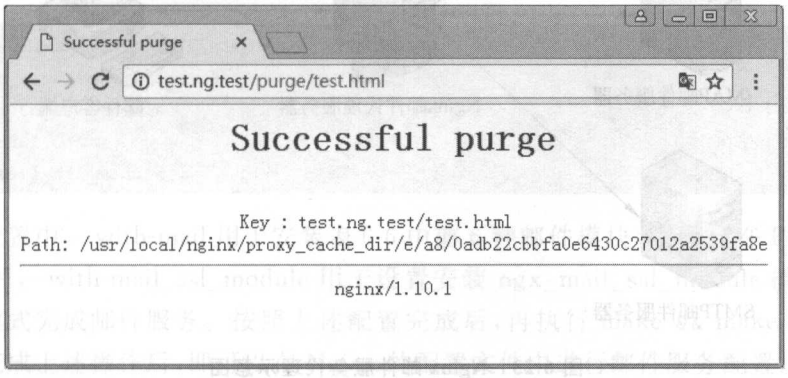


图 6-21　清理缓存文件

为了进一步验证是否成功清理了缓存文件，在缓存服务器中查看缓存目录，效果如图 6-22 所示，同时对比图 6-20，可以看出生成的缓存文件已经成功清除。

```
[root@localhost proxy_cache_dir]# tree
.
├── 7
│   └── 79
└── e
    └── a8

4 directories, 0 files
[root@localhost proxy_cache_dir]#
```

图 6-22　查看清理后的缓存文件目录

6.4　邮件服务

Nginx 除了上述讲解的常用功能外，还有一个基础的功能就是邮件代理服务，用于实现 Web 服务器与邮件服务器之间的代理，但是在实际应用中使用的并不多。因此，本节主要针对 Nginx 邮件服务的原理、涉及的指令以及相关协议进行讲解，用作读者对知识的扩展，不做重点介绍。

6.4.1 Nginx 实现邮件服务

Nginx 中提供的邮件模块，用于实现邮件代理服务器的功能，它不仅仅可以完成邮件服务的代理，同时还可以在 Nginx 与客户端、邮件服务器交互的过程中访问认证服务器，只有通过了认证，Nginx 才会在客户端用户与邮件服务器之间完成透传功能，如图 6-23 所示。

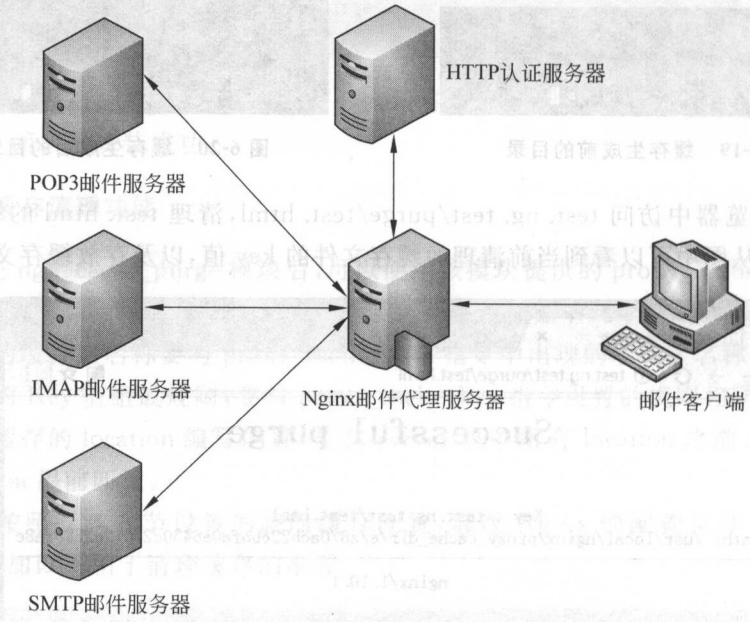


图 6-23 Nginx 邮件服务代理示意图

需要注意的是，Nginx 在与邮件客户端和邮件服务器进行传输的过程中，要遵循相应的邮件协议，常用的有 SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）、POP3（Post Office Protocol，邮局协议）和 IMAP（Internet Mail Access Protocol，互联网邮件访问协议）。其中，POP 协议现在普遍使用的是第三版，因此简称为 POP3。

3 种邮件协议的区别在于，SMTP 仅可以用于将邮件从发送方传输到目的方，而 POP3 和 IMAP 都可以完成邮件的接收，POP3 的客户端用户在读取邮件时需要全部下载才能进行操作，IMAP 可以通过客户端直接对服务器上的邮件进行操作，功能更加强大，但是缺点是占用的资源更多。在实际应用中具体采用哪种邮件协议要根据相应的需求进行合理设置。

6.4.2 邮件服务配置

Nginx 为邮件代理功能提供很多相关的模块，具体情况如表 6-6 所示。关于模块的概念会在第 7 章中详细讲解，此处读者了解即可。

在 Nginx 中，所有与邮件服务相关的配置都应该配置在 mail 块中，就像 Nginx 中所有与 Web 服务器相关的配置都设置在 http 块中一样。

表 6-6　邮件服务相关模块

模　块	说　明
ngx_mail_core_module	Nginx 邮件服务的核心功能
ngx_mail_auth_http_module	用于用户认证
ngx_mail_proxy_module	用于邮件代理
ngx_mail_smtp_module	用于 SMTP 邮件传输协议的相关设置
ngx_mail_pop3_module	用于 POP3 邮件传输协议的相关设置
ngx_mail_imap_module	用于 IMAP 邮件传输协议的相关设置
ngx_mail_ssl_module	支持基于 SSL/TLS 协议功能，要求 OpenSSL 库的支持

在邮件代理服务配置之前，首先需要按照前面讲解的方式，重新编译安装 Nginx，添加对邮件服务的支持，主要配置指令如下：

```
[root@localhost ~]#./configure \
--prefix=/usr/local/nginx \
--with-http_ssl_module \
--with-mail \
--with-mail_ssl_module
```

在上述配置中，--with-mail 用于安装表 6-6 中前 6 种邮件模块，表示允许 POP3、IMAP 和 SMTP 代理，--with-mail_ssl_module 用于设置安装 ngx_mail_ssl_module 模块，表示使用安全传输模式完成邮件服务。按照上述配置完成后，再执行 make && make install 重新编译安装。完成上述操作后，即可以在 Nginx 的配置文件中进行邮件服务配置。

关于如何搭建完整的邮件服务系统不是本章的讲解重点，下面介绍 Nginx 作为邮件代理服务器时最常用的配置方式。

```
1   mail {
2       #邮件认证服务器的访问 URL
3       auth_http  192.168.78.128:8009/auth.php;
4       #每个请求所使用的内存缓冲区大小
5       proxy_buffer 4k;
6       #添加对 POP3 的支持
7       server {
8           listen 110;
9           protocol pop3;
10          proxy on;
11      }
12      #添加对 IMAP 的支持
13      server {
14          listen 143;
15          protocol imap;
16          #设置接收初始客户端请求的缓冲区大小
17          imap_client_buffer 4k;
18          proxy on;
```

```
19      }
20      #添加对 SMTP 的支持
21      server {
22          listen 25;
23          protocol smtp;
24          proxy on;
25          #设置接收初始客户端请求的缓冲区大小
26          smtp_client_buffer 4k;
27          xclient off;
28      }
29  }
```

在上述设置中，第3行用于指定Nginx提供邮件代理服务时的HTTP认证服务器地址；第5行用来配置Nginx服务器代理缓存的大小；第7～28行配置，用于针对不同邮件协议(如POP3、IMAP、SMTP)进行设置。关于上述指令的具体含义说明如表6-7所示。

表6-7 邮件服务配置指令

指　令	说　明
listen	该指令用于配置邮件服务器监听的IP地址和端口
server_name	该指令用于为每个server块构成的虚拟主机配置的域名
protocol	用于配置当前虚拟主机支持的协议
so_keepalive	用于配置后端代理服务器是否启用TCP keepalive模式来处理Nginx邮件服务器转发的客户端连接
pop3_auth	用于配置POP3认证用户的方式
pop3_capabilities	用于配置POP3协议的扩展功能
imap_auth	用于配置IMAP认证用户的方式
imap_capabilities	用于配置IMAP协议的扩展功能
imap_client_buffer	用于配置IMAP协议数据缓存的大小
smtp_auth	用于配置SMTP认证用户的方式
smtp_capabilities	用于配置SMTP协议的扩展功能
auth_http_header	在Nginx服务器向HTTP认证服务器发起的认证请求时，添加指定的请求头
auth_http_timeout	配置Nginx服务器向HTTP认证服务器发起认证请求后，等待响应的超时时间
proxy_pass_error_message	用来配置是否将后端服务器上邮件服务认证过程中产生的错误信息发送给客户端
proxy_timeout	设置客户端与代理服务器之间的超时时间
xclient	可以开启或者关闭命令XCLIENT的SMTP后端连接，使得后端可以强制通过IP、HELO或LOGIN限定客户端

由于Nginx作为邮件代理服务器在实际应用中比较少。因此，这里只简单讲解其运行原理以及相关指令的含义，读者了解即可。若有兴趣，可以利用官方手册进行深入的学习。

本章小结

本章主要讲解 Nginx 的反向代理、负载均衡和 Web 缓存的配置。通过本章的学习，读者首先应该了解代理与反向代理的区别，负载均衡的作用以及 Web 缓存的实现原理。其次，能够熟练地配置出反向代理和负载均衡，并能够对需要缓存的网页设置合理的配置。同时，对于缓存的清理和邮件服务的配置要有一定的了解。

课后练习

一、填空题

1. 若要在 Nginx 中配置反向代理，需要使用_____指令指定后端服务器地址。
2. 将负载分摊到多个操作单元上执行，这种机制称为_____。
3. 在配置 Nginx 负载均衡时，若要使每个访客固定访问一个后端服务器，可以使用_____方式。
4. 在 proxy_cache_path 中，主动清空在 1 分钟内未被访问过的缓存的参数为_____。

二、判断题

1. 代理是指客户端通过代理服务器访问目标服务器。（　）
2. 在 Nginx 配置中，proxy_pass 指令用于配置缓存。（　）
3. Nginx 和邮件服务相关，用于用户认证的模块是 ngx_mail_proxy_module。（　）
4. 在配置 Nginx 负载均衡时，支持为每台后端服务器配置权重。（　）

三、选择题

在 Nginx 向 HTTP 认证服务器发起的认证请求时，添加指定的请求头使用（　）指令。

A. auth_http_header　　B. auth_http_timeout

C. imap_client_buffer　　D. protocol

四、操作题

除了本章讲解的两种缓存方式外，Nginx 还可以配合一些缓存软件完成相应的功能。Memcached 是一个高性能的分布式内存对象缓存系统，用于动态 Web 应用开发，以减轻数据库负载。接下来，请动手实现 Nginx＋Memcached 构建页面缓存的应用。

第7章 模块配置应用

学习目标

- 了解 Nginx 的模块化结构设计；
- 掌握 Nginx 的调试方法及手册的使用；
- 掌握网页压缩、重写、防盗链的配置；
- 熟悉 SSL、响应内容替换的配置。

通过前面的学习，读者应该对 Nginx 服务器的一些常见操作有了一定的认识。而这些操作涉及的相关指令，其实是 Nginx 中相关模块提供的功能，那么什么又是模块呢？本章将从模块应用的角度对 Nginx 进行详细介绍。

7.1 模块概述

7.1.1 模块化结构设计

Nginx 的内部结构是由核心部分和一系列功能模块组成的，正是由于 Nginx 的高度模块化设计，使得每个模块的功能相对简单，便于实现功能的扩展性。因此，在对 Nginx 模块讲解前，需要对模块化结构设计这种思想进行简单介绍。

对于到底什么是“模块化结构设计”，并没有一种统一的定义。在计算机中，最常见的说法就是，以功能块为单位进行程序设计，实现其求解算法的一种方法。模块化结构设计的目的是为了降低程序复杂度，使程序设计、调试和维护等操作简单化，方便团队协作以及应用的扩展升级。

换句话说，Nginx 就像是积木搭建的房子，在实现规范接口的前提下，各个团队只要保持接口不变，可同时开发功能模块，这样就可以根据实际需求不断地加入新的功能，或者去掉旧的功能，达到应用程序的高配置性、高扩展性、高定制性和高伸缩性。

7.1.2 Nginx 模块分类及作用

Nginx 从总体上来说是由许多个模块构成的，通常将 Nginx 分为 5 大模块，分别为核心模块、标准 HTTP 模块、可选 HTTP 模块、邮件服务模块和第三方模块。

其中，核心模块是 Nginx 服务器正常运行必不可少的模块，如同操作系统的内核。它提供了 Nginx 最基本的核心服务，例如前面章节讲解的权限控制、错误日志记录等；标准 HTTP 类型的模块用于支持标准 HTTP 的相关功能，是编译 Nginx 时默认安装的模块；可选 HTTP 模块主要用于扩展的 HTTP 功能，让 Nginx 能处理一些特殊的服务；邮件服务模

块主要用于支持邮件服务；第三方模块是为了扩展 Nginx 的应用，完成开发者想要的功能。

为了查看 Nginx 默认加载的核心模块和标准 HTTP 模块，切换到 Nginx 的解压目录中，打开 objs 目录下的 ngx_modules.c 文件。具体操作命令如下：

```
[root@localhost ~]#cd ~/nginx-1.10.1/objs
[root@localhost objs]#less ngx_modules.c
```

从 less 命令打开的文件，可以看到很多使用 extern 关键字定义的模块。具体如下所示。其中，第 1～3 行和第 5～8 行是核心模块，其余的都是 HTTP 标准模块。

```
1   extern ngx_module_t  ngx_core_module;
2   extern ngx_module_t  ngx_errlog_module;
3   extern ngx_module_t  ngx_conf_module;
4   extern ngx_module_t  ngx_openssl_module;
5   extern ngx_module_t  ngx_regex_module;
6   extern ngx_module_t  ngx_events_module;
7   extern ngx_module_t  ngx_event_core_module;
8   extern ngx_module_t  ngx_epoll_module;
9   extern ngx_module_t  ngx_http_module;
10  extern ngx_module_t  ngx_http_core_module;
11  extern ngx_module_t  ngx_http_log_module;
12  extern ngx_module_t  ngx_http_upstream_module;
13  extern ngx_module_t  ngx_http_static_module;
14  extern ngx_module_t  ngx_http_autoindex_module;
15  extern ngx_module_t  ngx_http_index_module;
16  extern ngx_module_t  ngx_http_auth_basic_module;
17  extern ngx_module_t  ngx_http_access_module;
18  extern ngx_module_t  ngx_http_limit_conn_module;
19  extern ngx_module_t  ngx_http_limit_req_module;
20  extern ngx_module_t  ngx_http_geo_module;
21  extern ngx_module_t  ngx_http_map_module;
22  extern ngx_module_t  ngx_http_split_clients_module;
23  extern ngx_module_t  ngx_http_referer_module;
24  extern ngx_module_t  ngx_http_rewrite_module;
25  extern ngx_module_t  ngx_http_ssl_module;
26  extern ngx_module_t  ngx_http_proxy_module;
27  extern ngx_module_t  ngx_http_fastcgi_module;
28  extern ngx_module_t  ngx_http_uwsgi_module;
29  extern ngx_module_t  ngx_http_scgi_module;
30  extern ngx_module_t  ngx_http_memcached_module;
31  extern ngx_module_t  ngx_http_empty_gif_module;
32  extern ngx_module_t  ngx_http_browser_module;
33  extern ngx_module_t  ngx_http_secure_link_module;
34  extern ngx_module_t  ngx_http_upstream_hash_module;
35  extern ngx_module_t  ngx_http_upstream_ip_hash_module;
36  extern ngx_module_t  ngx_http_upstream_least_conn_module;
37  extern ngx_module_t  ngx_http_upstream_keepalive_module;
38  extern ngx_module_t  ngx_http_upstream_zone_module;
39  extern ngx_module_t  ngx_http_stub_status_module;
40  extern ngx_module_t  ngx_http_write_filter_module;
```

```
41  extern ngx_module_t  ngx_http_header_filter_module;
42  extern ngx_module_t  ngx_http_chunked_filter_module;
43  extern ngx_module_t  ngx_http_range_header_filter_module;
44  extern ngx_module_t  ngx_http_gzip_filter_module;
45  extern ngx_module_t  ngx_http_postpone_filter_module;
46  extern ngx_module_t  ngx_http_ssi_filter_module;
47  extern ngx_module_t  ngx_http_charset_filter_module;
48  extern ngx_module_t  ngx_http_sub_filter_module;
49  extern ngx_module_t  ngx_http_userid_filter_module;
50  extern ngx_module_t  ngx_http_headers_filter_module;
51  extern ngx_module_t  ngx_http_echo_module;
52  extern ngx_module_t  ngx_http_copy_filter_module;
53  extern ngx_module_t  ngx_http_range_body_filter_module;
54  extern ngx_module_t  ngx_http_not_modified_filter_module;
```

从上述列出的模块不难看出，Nginx 的模块命名规则，是以 ngx_开头，以_module 结尾，中间使用一个或多个英文单词描述该模块的功能。例如，ngx_core_module 表示该模块提供的是 Nginx 的核心功能。

7.1.3 Nginx 手册的使用

Nginx 的模块有很多，与其相关的指令和内置变量也特别多。因此，在开发中掌握如何查看 Nginx 官方提供的文档，会起到事半功倍的效果。

在 Nginx 官方网站 http://nginx.org 中，单击右侧栏中的 documentation 链接切换到手册页面。如图 7-1 所示。该文档是由 Introduction（介绍）、How-To（操作指南）、Development（发展史）以及 Modules reference（模块参考指南）4 部分组成。其中，Modules reference 中提供了 Nginx 所有模块以及指令的使用，具体如图 7-2 所示。

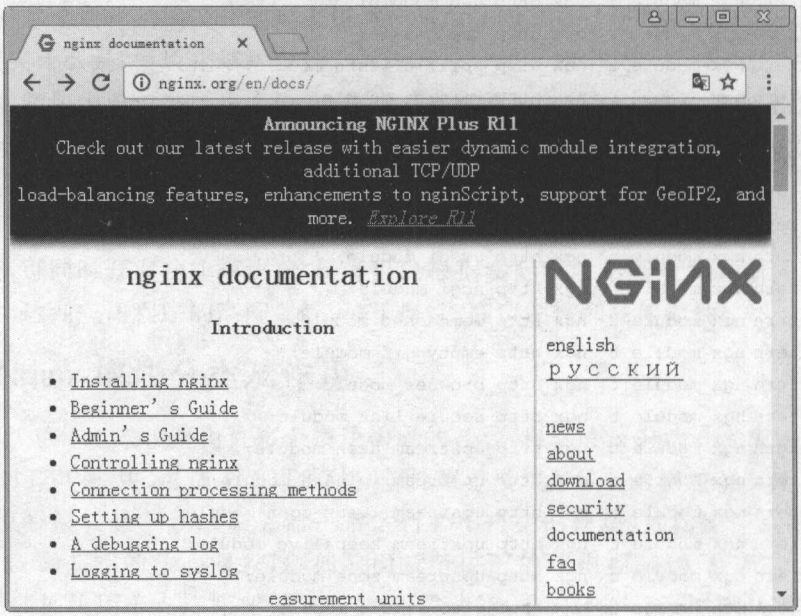


图 7-1 官网手册首页

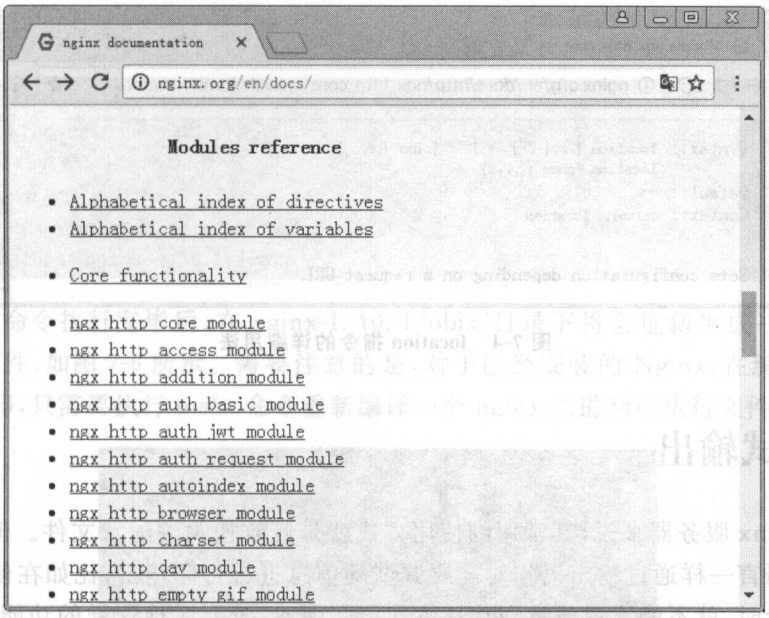


图 7-2　模块参考指南

在图 7-2 中，Alphabetical index of directives 表示按照字母顺序排序的所有 Nginx 指令，Alphabetical index of variables 表示按照字母顺序排序的所有 Nginx 内置变量，Core functionality 中提供了 Nginx 的核心功能，其他的则是 Nginx 提供的几大服务（http、mail、stream），每个服务的第一个都是其核心功能块，剩余的模块按照字母顺序进行排列。

在指令和变量名称的后面，若标注了具体的模块名称，即表示该指令或变量只能在此模块下使用，未标注的则表示通用指令和变量。

以查找前面学习过的 location 指令为例，首先单击 Alphabetical index of directives，然后按 Ctrl+F 键，在浏览器的搜索框中输入要查找的指令 location，回车后将看到如图 7-3 所示的效果。

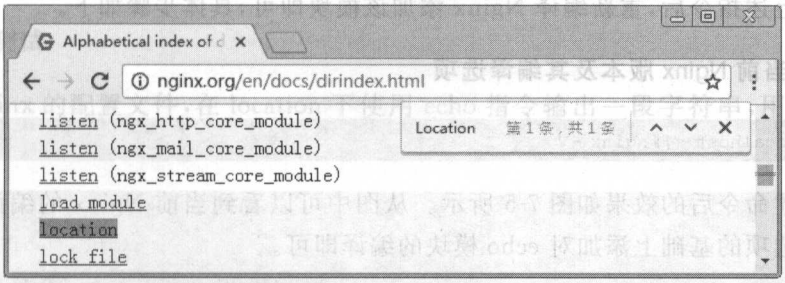


图 7-3　查看 location 指令

接着单击 location 指令的链接，查看其具体的使用，如图 7-4 所示。从图中可以看到 location 的语法、默认值、可以编写的位置，以及该指令的解释说明、示例和注意事项等具体内容。

在学习的过程中，学会如何查看手册，将对 Nginx 的学习和理解起到重要的辅助作用。

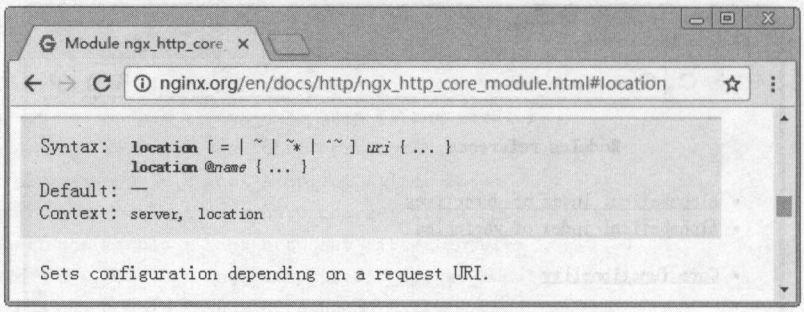


图 7-4 location 指令的详细用法

7.2 调试输出

对于 Nginx 服务器来说，其常用的使用方式就是修改相关的配置文件。但是它本身又不能像编程语言一样通过输出语句对需要调试的信息进行打印输出，比如在使用 Nginx 的一些内置变量时，就不能直观地看到其具体的值。因此，若要实现这样的功能，可以使用第三方提供的 echo-nginx-module 模块，下面将对该模块的配置与相关指令的使用进行详细介绍。

7.2.1 调试输出的配置

由于 echo-nginx-module 模块是第三方提供的模块，首先需要从网上下载放到指定目录中。从 GitHub 平台获取模块的下载地址，具体操作如下。

```
[root@localhost ~]#wget \
https://github.com/openresty/echo-nginx-module/archive/v0.60.tar.gz
[root@localhost ~]#tar -zxvf v0.60.tar.gz
[root@localhost ~]#mv  echo-nginx-module-0.60 echo-nginx-module
```

执行完上述指令后，重新编译 Nginx 添加该模块即可，具体步骤如下。

1. 查看当前 Nginx 版本及其编译选项

```
[root@localhost ~]#nginx -V
```

执行上述命令后的效果如图 7-5 所示。从图中可以看到当前 Nginx 的编译选项，接下来在原编译选项的基础上添加对 echo 模块的编译即可。

```
[root@localhost ~]# nginx -V
nginx version: nginx/1.10.1
built by gcc 4.4.7 20120313 (Red Hat 4.4.7-17) (GCC)
built with OpenSSL 1.0.1e-fips 11 Feb 2013
TLS SNI support enabled
configure arguments: --prefix=/usr/local/nginx --with-http_ssl_module
[root@localhost ~]#
```

图 7-5 查看 Nginx 版本及编译选项

2. 重新编译 Nginx

```
[root@localhost ~]#cd nginx-1.10.1
[root@localhost nginx-1.10.1]#./configure \
--prefix=/usr/local/nginx \
--with-http_ssl_module \
--add-module=/root/echo-nginx-module
[root@localhost nginx-1.10.1]#make
```

按照上述命令执行完毕后，在 nginx-1.10.1/objs/目录下将会重新生成一个 nginx 二进制可执行文件，如图 7-6 所示。需要注意的是，对于已经安装的 Nginx，在编译时不再需要 make install，只需要执行 make 命令重新编译一个 nginx 二进制可执行文件即可。

```
[root@localhost objs]# ll
total 6332
drwxr-xr-x. 3 root root    4096 Oct 13 11:56 addon
-rw-r--r--. 1 root root   15363 Oct 26 14:47 autoconf.err
-rw-r--r--. 1 root root   44482 Oct 26 14:47 Makefile
-rwxr-xr-x. 1 root root 6289322 Oct 26 14:48 nginx
-rw-r--r--. 1 root root    5341 Oct 26 14:48 nginx.8
-rw-r--r--. 1 root root    6683 Oct 26 14:47 ngx_auto_config.h
-rw-r--r--. 1 root root     657 Oct 26 14:47 ngx_auto_headers.h
-rw-r--r--. 1 root root    5793 Oct 26 14:47 ngx_modules.c
-rw-r--r--. 1 root root   91592 Oct 26 14:48 ngx_modules.o
drwxr-xr-x. 9 root root    4096 Oct 13 11:24 src
[root@localhost objs]#
```

图 7-6　重新编译后生成的 nginx 可执行文件

3. 备份并复制 nginx 的可执行文件

```
[root@localhost nginx-1.10.1]#cd objs
[root@localhost objs]#mv /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.bak
[root@localhost objs]#cp nginx /usr/local/nginx/sbin/nginx
```

上述第 2 行指令，用于备份原 nginx 可执行文件为 nginx.bak，第 3 行用于将新生成的 nginx 可执行文件复制到 Nginx 安装目录下的 sbin 目录中。

4. 验证测试

打开 Nginx 的配置文件，在 location 下使用 echo 指令输出一段字符串，用于测试。具体配置如下：

```
1    location / {
2        root   html;
3        index  index.html index.htm;
4        default_type text/plain;
5        echo "This is an echo module.";
6    }
```

上述第 4 行设置中，default_type 指令用于指定 MIME 类型，这里将其设置为文本格式；echo 指令用于输出其后的内容。按照上述设置，完成修改 nginx.conf 文件后，平滑重启 Nginx 使配置生效。

访问测试如图 7-7 所示。若能看到如图 7-7 所示的效果，证明 echo 模块添加成功，在 Nginx 配置文件中就可以使用 echo 指令进行调试了。

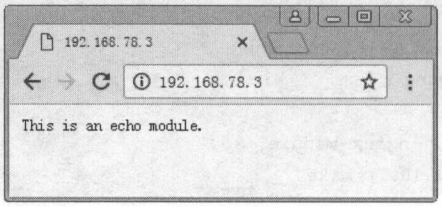


图 7-7 echo 测试

7.2.2 常见的应用案例

在第三方模块 echo-nginx-module 中最常用的就是 echo 指令，且该指令只能在 location 块或 location 块下 if 指令中使用。下面演示几种 echo 指令常见的用法，具体如下。

1. 普通输出

```
1    location / {
2        default_type text/plain;
3        echo 45 * 78;
4        echo this is a test;
5        echo $args;
6    }
```

在上述配置中，第 3、4 行用于输出字符串，第 5 行 $args 内置变量用于输出 HTTP 请求时传递的参数。重启 Nginx 使配置生效，如图 7-8 所示。

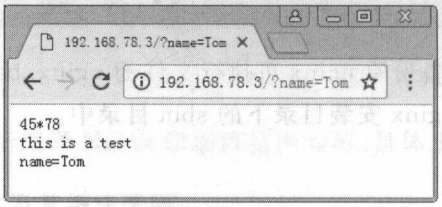


图 7-8 普通输出

从图 7-8 中可以看出，echo 指令后的内容可以是一个字符串或是 Nginx 的内置变量。且当 MIME 为文本类型时，echo 指令的每一次执行后都有一个换行。

2. 带参数输出

```
1    location / {
2        default_type text/plain;
3        echo -n Hello, ;
4        echo itheima;
5    }
```

若要想在 echo 指令输出后不换行，可以像上述第 3 行配置一样，添加一个-n 参数。平滑重启 Nginx 使配置生效，然后在浏览器中访问，效果如图 7-9 所示。

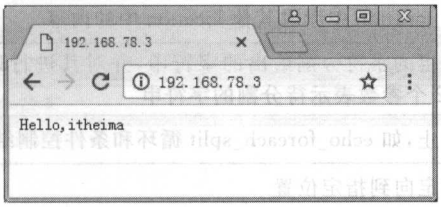


图 7-9　带参数输出

3. 输出特殊字符

```
1   location / {
2       default_type text/plain;
3       echo \"welcome to China\";
4       echo \'welcome to China\';
5       echo \\n and \\t is Special characters;
6       echo -- -n is an option;
7       echo -- ----is the separator;
8   }
```

在使用 echo 输出内容时，若要同时输出特殊字符，如双引号、单引号、换行符、制表符等时，可以使用转义字符，如第 3～5 行中使用斜线“\”进行转换；若想要输出横杠“-”时，则需要使用双横杠“--”对其转义，如第 6～7 行配置。对于 Nginx 来说，$ 是一个特殊的符号，echo 指令对其不能进行输出。平滑重启 Nginx 使配置生效，如图 7-10 所示。

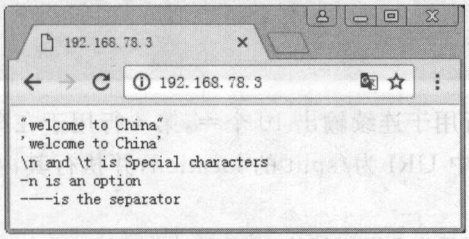


图 7-10　输出特殊字符

对于 echo 模块，除了上述提到的 echo 指令外，还有很多其他常用的指令，如表 7-1 所示。

表 7-1　echo 模块其他常用指令

指　令	说　明
echo_duplicate	按照指定的次数重复输出指定内容，第 1 个参数为次数，第 2 个参数为指定内容
echo_flush	刷新缓冲区的内容，并输出
echo_sleep	按照指定的秒数，延迟输出
echo_reset_timer	重置当前请求花费的时间

续表

指　令	说　明
echo_location	在当前 location 中读取其他 location 中的内容
echo_foreach_split	按照指定的字符分割给出的字符串，并对其进行遍历，其第 1 个参数表示分隔符，第 2 个参数表示待分割的字符串
echo_end	用于终止，如 echo_foreach_split 循环和条件控制结构
echo_exec	内部重定向到指定位置
echo_status	指定默认的响应状态码
echo_before_body	在输出过滤器中整体内容输出前，输出指定内容
echo_after_body	在输出过滤器中整体内容输出后，输出指定内容

为了使读者能够熟练地掌握 echo 模块这些常用指令的使用，下面以综合案例的形式进行演示。常见的几种使用组合方式，具体如下。

1）循环遍历

```
1   location / {
2       default_type text/plain;
3       echo_duplicate 10 "=";
4       echo;
5       echo_location /split;
6       echo_status 404;
7   }
8   location /split {
9       echo_foreach_split "," "PHP,Nginx,C/C++,Java,IOS";
10      echo $echo_it;
11      echo_end;
12  }
```

在上述配置中，第 3 行用于连续输出 10 个＝，第 4 行用于在浏览器中输出时换行，第 5 行用于匹配当前配置文件中 URI 为/split 的 location，并执行其内的语句，第 6 行用于指定响应状态码。

其中，第 9 行表示以逗号“,”分割给定的字符串"PHP,Nginx,C/C＋＋,Java,IOS"，并循环遍历分割后的结果，echo 模块的内置变量 $echo_it 表示每次循环的值，echo_end 指令用于终止 echo_foreach_split 指令的执行。接下来，重启 Nginx 使配置生效，可在浏览器中验证测试，如图 7-11 所示。

2）延迟后，重置请求时间

```
1   location / {
2       default_type text/plain;
3       echo_sleep 3;
4       echo "$echo_timer_elapsed s";
5       echo_reset_timer;
6       echo_sleep 2;
```

```
7        echo "$echo_timer_elapsed s";
8    }
```

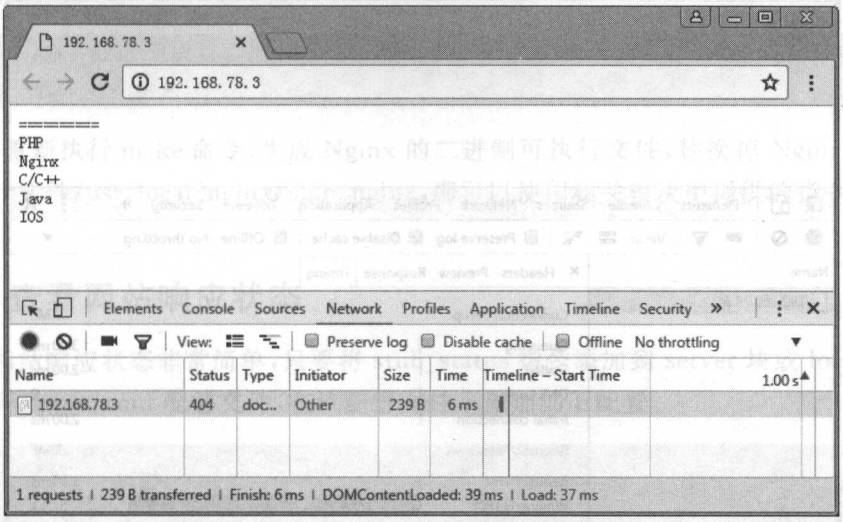


图 7-11　循环遍历

在上述配置中，第 3 行用于指定当前请求的延迟时间为 3 秒，第 4 行利用 echo 模块中的内置变量 $echo_timer_elapsed 获取从 HTTP 请求开始到当前时间的秒数，第 5 行用于重置当前 HTTP 请求的时间。

接下来，平滑重启 Nginx 使配置生效，在浏览器中访问。为了直观地看到整个请求使用的时间，按 F12 键打开开发者工具，单击当前的请求，在标签栏中选择 Timing，查看 HTTP 完成请求花费的时间以及延迟等待的时间，如图 7-12 所示。

3）指定输出的前后内容

```
1    location / {
2        default_type text/plain;
3        echo_exec @proxy;
4    }
5    location @proxy {
6        echo_before_body start;
7        echo_after_body end;
8        proxy_pass http://192.168.78.128;
9    }
```

上述第 3 行配置，用于在 Nginx 内部重定向到名称为 proxy 的 location 处执行。其中，第 6 行用于指定在反向代理请求的内容输出前展示的内容，第 7 行用于指定在反向代理请求的内容输出后展示的内容。

这里假设在 IP 为 192.168.78.128 的 Web 服务器网站根目录下，有一个名为 index.html 的默认访问文件，该文件中的内容具体如下。

```
<h1>web server:192.168.78.128!</h1>
```

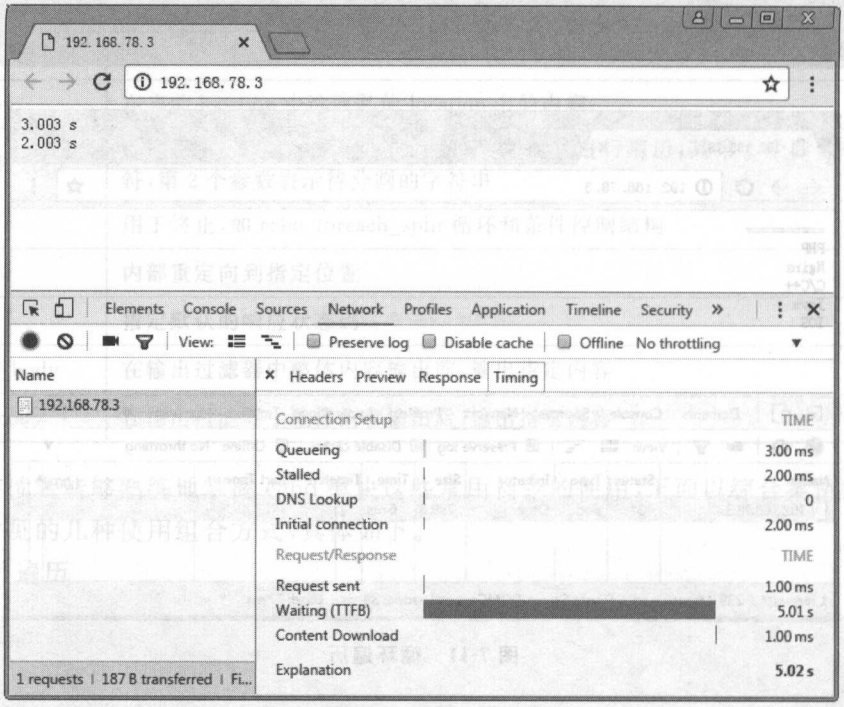


图 7-12 延迟后重置请求的时间

准备完成后，在浏览器中访问测试的效果如图 7-13 所示。从图 7-13 中可以看出，start 按照指定的格式出现在 web server:192.168.78.128!前，end 按照指定的格式出现在 web server:192.168.78.128!之后。

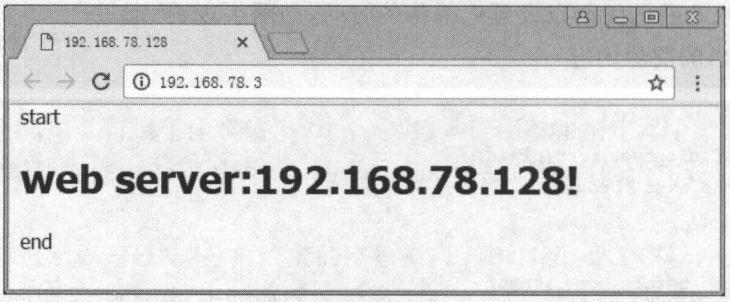


图 7-13 指定输出的前后内容

7.3 查看响应状态与替换响应内容

7.3.1 安装所需模块

对于一个网站来说，若想要查看当前网站的响应状态、修改响应内容中的敏感词汇或是临时想要在网站中添加一个通用的 CSS 文件等，则可以使用 Nginx 服务器提供的 ngx_http_stub_status_module 和 ngx_http_sub_module 模块。

由于在 nginx 默认配置编译安装时，未添加对 ngx_http_stub_status_module 和 ngx_http_sub_module 模块的支持，最简便的做法就是利用 7.2.1 节讲解的方式在./configure 时添加以下配置。

```
--with-http_stub_status_module \
--with-http_sub_module
```

然后，重新执行 make 命令，生成 Nginx 的二进制可执行文件，替换掉 Nginx 安装目录中的可执行文件/usr/local/nginx/sbin/nginx，即可以使用相关模块中提供的指令进行具体配置。

7.3.2　查看网站响应状态

查看网站响应状态非常简单，只要将 stub_status 指令添加到 server 块或 location 块下即可。打开 nginx.conf 配置文件，在该配置文件下添加如下配置。

```
server {
    listen 80;
    server_name www.test.com;
    root html/test.com;
    index index.html index.htm;
    stub_status;
}
```

平滑重启 Nginx 使配置生效，在浏览器中访问 http://www.test.com 的效果如图 7-14 所示。

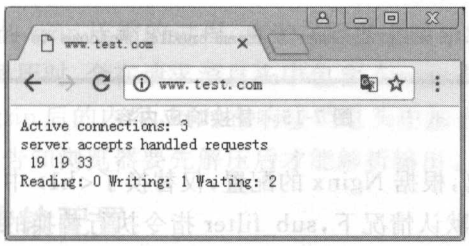


图 7-14　查看网站响应内容

从图 7-14 中可以看出，当前显示的状态数据包含 7 部分内容，分别为 Active connections(活跃连接数量)、server accepts(服务器处理连接数)、handled(服务器创建的握手次数)、requests(服务器处理请求连接的数量)、Reading(读取客户端的连接数)、Writing(响应数据到客户端的数量)以及 Waiting(正在等候下一次请求指令的驻留连接数)。

7.3.3　替换网站响应内容

打开 nginx.conf 配置文件，在该配置文件下添加以下配置。

```
server {
    listen 80;
```

```
3        server_name www.test.com;
4        root html/test.com;
5        index  index.html index.htm;
6        sub_filter  itheima '黑马程序员';
7    }
```

上述第 6 行中的 sub_filter 指令，用于将响应内容中的 itheima 替换为中文的“黑马程序员”。为了查看到响应效果，接下来在域名为 www. test. com 的网站目录下编写 index. html 文件，具体如下。

```
1    <!DOCTYPE html>
2    <html>
3        <head><title>Welcome</title><meta charset="utf-8"></head>
4        <body>
5            <h1>Welcome to itheima!</h1>
6            <p>itheima is the programmer's dream cradle,Welcome to itheima!</p>
7        </body>
8    </html>
```

从上述网页内容可以看出，itheima 在<h1>标签和<p>标签中各出现了一次。在浏览器中访问 http://www. test. com/index. html 的效果如图 7-15 所示。

图 7-15　替换响应内容

从图 7-15 中可以看出，根据 Nginx 的配置，仅替换了<h1>中的 itheima，<p>标签中内容并未被替换。这是由于默认情况下，sub_filter 指令执行替换的次数由 sub_filter_once 指令决定，默认情况下 sub_filter_once 指令的值设置为 on，表示仅替换一次。

修改上述的 Nginx 配置文件，添加如下指令，允许替换响应内容中全部出现的 itheima。

```
sub_filter_once off;
```

修改完成后，重新访问的效果如图 7-16 所示。

除了上述提到的指令外，在替换网站响应内容时，还可以使用 sub_filter_types 指令设置需要被替换的响应内容 MIME 类型，其默认值为 text/html，因此，若在配置文件中再次指定 text/html，Nginx 服务启动时会出现重复 MIME 类型的警告提示。另外，sub_filter_types 指令还有一个特殊的值 *，表示所有 MIME 类型。

图 7-16　替换所有相关响应内容

7.4　网页压缩传输

7.4.1　gzip 压缩技术

gzip(GNU-ZIP)是一种压缩技术，经过 gzip 压缩后，页面大小可以变为原来的 30%甚至更小。这样，用户浏览页面的时候速度会快得多。gzip 网页压缩的实现需要浏览器和服务器的支持，如图 7-17 所示。

图 7-17　gzip 压缩过程

从图 7-17 中可以看出，gzip 压缩的过程，首先在服务器端压缩，然后传到浏览器端后解压。当浏览器支持 gzip 解压时，会在请求消息头中包含 Accept-Encoding：gzip，这样 Nginx 就会向浏览器发送经过 gzip 后的内容，同时在响应消息头中加入 Content-Encoding：gzip，声明这是 gzip 后的内容，告知浏览器要先解压后才能解析输出。

7.4.2　网页压缩传输配置

Nginx 服务器为网页压缩专门提供了 gzip 模块，并且模块中的相关指令均可以设置在 http、server 或 location 块中，实现服务器端按照指定的设置进行压缩。具体指令及其含义，如表 7-2 所示。

表 7-2　gzip 模块中相关配置指令

指　令	说　明
gzip	该指令用于开启或关闭 gzip 模块
gzip_buffers	设置系统获取几个单位的缓存用于存储 gzip 的压缩结果数据流
gzip_comp_level	gzip 压缩比，压缩级别是 1～9，1 的压缩级别最低，9 的压缩级别最高。压缩级别越高压缩率越大，压缩时间越长
gzip_disable	可以通过该指令对一些特定的 User-Agent 不使用压缩功能

续表

指　令	说　明
gzip_min_length	设置允许压缩的页面最小字节数，页面字节数从响应消息头的 Content-Length 中进行获取
gzip_http_version	识别 HTTP 协议版本，其值可以是 1.1(默认值)或 1.0
gzip_proxied	用于设置启用或禁用从代理服务器上收到响应内容的 gzip 压缩功能
gzip_types	匹配 MIME 类型进行压缩。且无论是否指定，text/html 类型总是会被压缩的
gzip_vary	用于在响应消息头中添加 Vary：Accept-Encoding，使代理服务器根据请求头中的 Accept-Encoding 识别是否启用 gzip 压缩。

在了解 gzip 模块的指令后，还需要大量的综合练习，才能够灵活熟练地运用。接下来演示 gzip 的配置案例，来帮助读者进一步学习、理解和应用。

1. 修改配置文件

打开 nginx.conf 配置文件，在 http 块中添加以下配置，用于完成网页压缩输出功能。

```
1   http {
2       ...
3       gzip on;
4       gzip_types text/plain application/javascript text/css;
5       ...
6   }
```

在上述设置中，第 3 行用于启用 gzip 模块，第 4 行用于在客户端访问网页时，对文本、JavaScript 和 CSS 文件进行压缩输出。

2. 访问测试

平滑重启 Nginx，使配置生效。在浏览器中访问测试，按 F12 键打开开发者工具，单击当前的请求，在标签栏中选择 Headers，查看 HTTP 响应头信息，如图 7-18 所示。

从图 7-18 中可以看出，当前 Content-Encoding(内容编码)为 gzip 类型，Content-Type(内容类型)为 HTML 网页类型，Transfer-Encoding(传输编码)为 chunked(块编码)表示内容长度不确定。

接下来，关闭 gzip 模块，查看 HTTP 响应头信息，如图 7-19 所示。从图 7-19 中可以看到，当前的 Content-Length(内容大小)为 612 字节。

除了上述的基本配置外，在实际开发中还可以具体配置压缩比，缓存大小、对于代理是否采用压缩等详细的设置，具体示例如下。

```
1   gzip_buffers     4 16k;
2   gzip_comp_level 4;
3   gzip_disable "MSIE [1-6].";
4   gzip_min_length  5k;
5   gzip_http_version 1.0;
6   gzip_proxied any;
7   gzip_vary on;
```

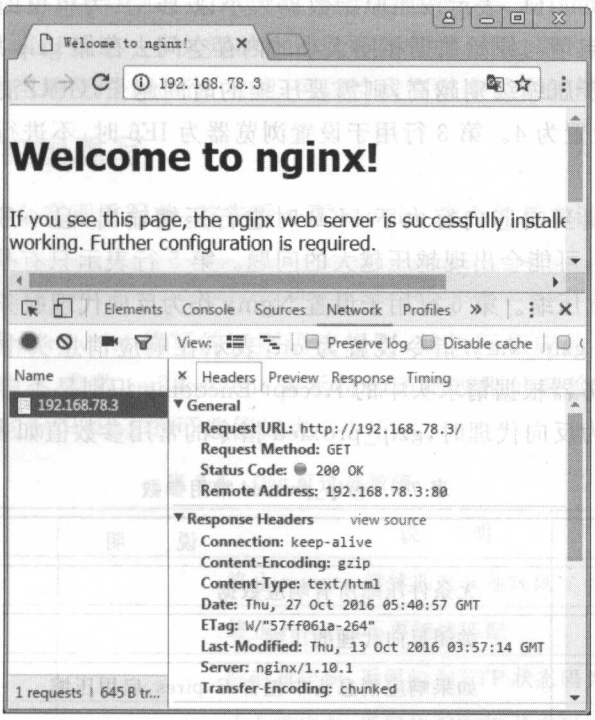


图 7-18　开启 gzip 模块

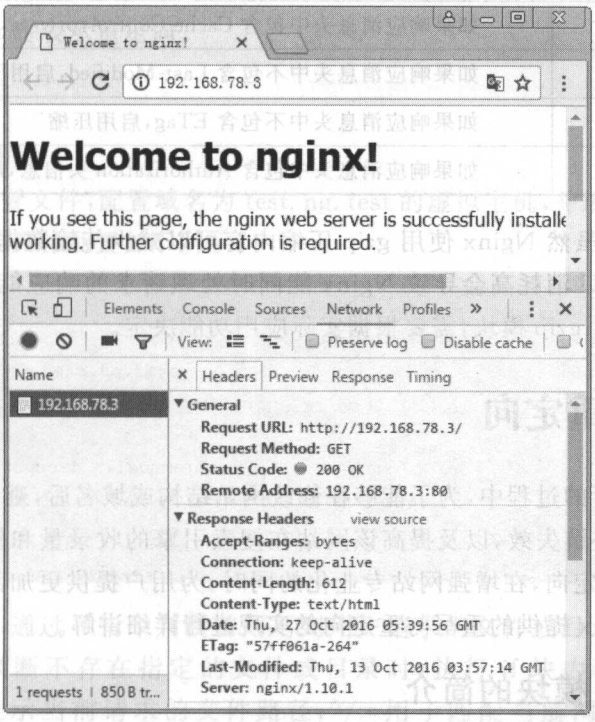


图 7-19　关闭 gzip 模块

上述第1行中的4 16k，表示按照原始数据大小以16KB为单位的4倍申请内存；如果没有设置，默认值是申请与原始数据相同大小的内存空间去存储gzip压缩结果。第2行表示压缩级别为4，由于压缩级别越高，则需要压缩的时间越长，CPU消耗也越大。因此，一般推荐将压缩级别设置为4。第3行用于设置浏览器为IE6时，不进行压缩，防止出现页面假死的现象。

第4行用于设置当响应内容大于5KB时进行压缩输出，且一般建议最小值设置为1KB，当小于1KB时，可能会出现越压越大的问题。第5行表示只有是HTTP/1.0协议的请求时才会进行gzip压缩。第6行用于设置Nginx作为反向代理服务器时，无条件压缩所有结果数据，第7行gzip_vary指令设置为on，表示在响应消息头中添加Vary：Accept-Encoding，使代理服务器根据请求头中的Accept-Encoding识别是否启用Gzip压缩。

其中，Nginx作为反向代理时，gzip_proxied指令的常用参数值如表7-3所示。

表7-3 gzip_proxied常用参数

指　令	说　明
any	无条件压缩所有响应数据
off	关闭反向代理的压缩
expired	如果响应消息头中包含Expires，启用压缩
no-cache	如果响应消息头中包含Cache-Control:no-cache，启用压缩
no-store	如果响应消息头中包含Cache-Control:no-store，启用压缩
private	如果响应消息头中包含Cache-Control:private，启用压缩
no_last_modified	如果响应消息头中不包含Last-Modified，启用压缩
no_etag	如果响应消息头中不包含ETag，启用压缩
auth	如果响应消息头中包含Authorization头信息，启用压缩

值得一提的是，虽然Nginx使用gzip压缩内容可以减少传输数据大小，但压缩率越大CPU消耗越高。CPU消耗高会导致Nginx能同时处理请求的响应能力下降。因此，在实际应用中是否要开启gzip模块，需要根据实际应用功能决定。

7.5 重写与重定向

在实际网站运营的过程中，为了能够在修改网站结构或域名后，避免造成网站中的链接或在其他网站中的外链失效，以及提高该网站在搜索引擎的收录量和排名等目的。通常会采用URL重写与重定向，在增强网站专业化的同时，为用户提供更加舒适的使用体验。接下来，本节将对Nginx提供的重写与重定向的实现进行详细讲解。

7.5.1 rewrite模块的简介

重写与重定向功能是现在大多数Web服务器都支持的一项功能，相对于其他产品而言，Nginx中的rewrite模块提供的功能在配置上更加的灵活自由，可定制性非常的高。它

的实现方式也非常的简单，只需要通过 rewrite 指令根据 Nginx 提供的全局变量或自定义的变量，结合正则表达式以及进一步处理的标识就可以完成 URL 重写或重定向。

因此，在网站中适当地使用 rewrite 功能，可以给我们带来很多的便利。

7.5.2　rewrite 实现重写

在使用 rewrite 指令实现重写前，首先看一下 rewrite 指令的基本语法，具体如下：

```
rewrite regex replacement [flag];
```

上述语法表示，符合 rewrite 编写的正则语法规则，就执行相应的替换算法。其中，参数 regex 表示正则表达式，参数 replacement 表示符合正则规则的替换算法，可选参数 flag 用于指定进一步处理的标识，flag 的可选值如表 7-4 所示。

表 7-4　flag 可选参数值

参　数　值	说　　明
last	终止 rewrite，继续匹配其他规则
break	终止 rewrite，不再继续匹配
redirect	临时重定向，返回的 HTTP 状态码为 302
permanent	永久重定向，返回的 HTTP 状态码为 301

在表 7-4 中，当 flag 的值为 last 或 break 时，表示当前的设置为重写，当 flag 的值为 redirect 或 permanent 时表示重定向。

1. rewrite 的重写

1）添加 rewrite 指令

打开 Nginx 的配置文件，配置域名为 test.ng.test 的虚拟主机，并规定访问不到文件或目录时执行重写操作，具体配置如下所示。

```
1  server {
2      listen 80;
3      server_name test.ng.test;
4      index index.html index.htm;
5       root html;
6      if(!-e $request_filename){
7          rewrite "^/.*" /default/default.html break;
8      }
9  }
```

上述第 6 行配置，通过 if 指令判断访问不到用户请求的文件或目录时，执行第 7 行指令。其中，!-e 用于判断不存在指定的文件或目录时，执行 if 块内的语句。内置变量 $request_filename 表示当前请求的文件路径；^/.*用于匹配当前网站下的所有请求，/default/default.html 用于替换符合指定规则的请求。

值得一提的是，if 指令根据给定的条件进行判断，如果判断结果为 true，则执行大括号

“{}”内的指令。当判断条件仅是一个变量时，如果值为空或任何以 0 开头的字符串都会当做 false，不再执行大括号内的指令。

if 指令中可以使用的判断符号如表 7-5 所示。

表 7-5 if 指令判断符号

判断符号	说 明	判断符号	说 明
=	判断变量与内容相等	!=	判断变量与内容不等
~	区分大小写正则匹配	~*	不区分大小写正则匹配
!~	区分大小写正则不匹配	!~*	不区分大小写正则不匹配
-f	判断文件存在	!-f	判断文件不存在
-d	判断目录存在	!-d	判断目录不存在
-e	判断文件或目录存在	!-e	判断文件或目录不存在
-x	判断可执行文件	!-x	判断不可执行文件

2）测试验证

在网站根目录下创建 default 目录，并在该目录下编写 default.html 文件，具体内容如下。

```
<h1>Welcome to html/default/default.html!</h1>
```

可在浏览器中访问一个不存在的文件，如 http://test.ng.test/test/demo.html，如图 7-20 所示。从图 7-20 中可以看到，浏览器的 URL 请求地址不变，Nginx 仅按照定义的重写规则，将 html/default/default.html 文件中的内容重写到当前的 HTTP 请求中。

图 7-20 rewrite 重定向

2. break 和 last 标识的区别

在使用 rewrite 实现重写时，需要注意 flag 可选参数值 break 和 last 的区别，前者在 rewrite 指令匹配成功后就不再进行匹配，而后者在 rewrite 后会根据 rewrite 匹配的规则重新发起一个请求继续进行匹配。为了明确地看到两者的区别，下面通过一个案例进行验证。

1）设置匹配规则

```
1    server {
```

```
2        listen 80;
3        server_name test.ng.test;
4        root html;
5        location /break/ {
6            rewrite ^/break/(.*)/test/$1 break;
7            echo "break page";
8        }
9        location /last/ {
10           rewrite ^/last/(.*)/test/$1 last;
11           echo "last page";
12       }
13       location /test/ {
14           echo "test page";
15       }
16   }
```

上述配置，根据 location 的不同匹配规则，执行不同的 rewrite 重写，并通过 echo 指令在 rewrite 后输出不同的提示信息。

2）测试验证

在浏览器中访问 http://test.ng.test/break/break.html 的效果如图 7-21 所示。接着在浏览器中访问 http://test.ng.test/last/last.html，如图 7-22 所示。

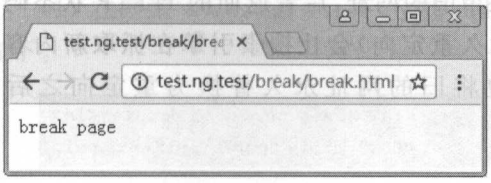


图 7-21　break 访问结果

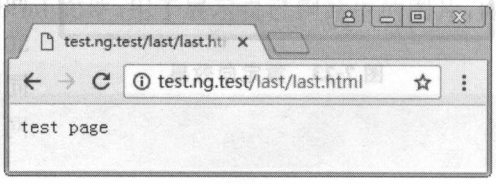


图 7-22　last 访问结果

对比上述验证结果，当用户的请求符合第 5 行 location 规则时，执行第 6 行 rewrite 指令后，根据 break 的指定，继续执行第 7 行配置。而当用户的请求符合第 9 行 location 规则时，执行第 10 行 rewrite 指令后，根据 last 的指定，按照匹配到的替换算法发起一个新的请求 http://test.ng.test/test/last.html，最终匹配到第 13 行的 location 规则，然后执行第 14 行的 echo 输出语句。

因此，在实际使用 rewrite 配置重写时，要根据实际情况选择合适的 flag 可选参数值，否则会造成与预期不一样的结果。

7.5.3　rewrite 实现重定向

rewrite 的重定向就是将用户访问的 URL 修改为重定向的地址，只需将 flag 的可选参数值设置为 redirect 或 permanent 即可实现。具体配置如下。

```
1    server {
2        listen 80;
3        server_name test.ng.test;
4        root html;
```

```
5        set $name $1;
6        rewrite ^/img-([0-9]+).jpg$ /img/$name.jpg permanent;
7    }
```

上述第 5 行配置，利用 set 指令为变量 $name 赋值，$1 表示符合正则表达式第一个子模式的值，如第 6 行中的子模式([0-9]+)匹配到的值，可以是 2、45 等由一个或多个数字组成的字符串。第 6 行用于在用户请求“http://test.ng.test/img-数字.jpg”时，重定向到“http://test.ng.test/img/数字.jpg”。

接下来，在 test.ng.test 的网站目录中创建一个用于存放图片的 img 目录，然后在该目录中保存一个文件名为 2.jpg 的图片。为了测试当前配置是否成功，通过浏览器访问 http://test.ng.test/img-2.jpg，运行结果如图 7-23 所示。从图 7-23 中可以看出，用户发出的 URL 请求只要符合 rewrite 定义的正则规则，就会按照其后的替换规则执行。

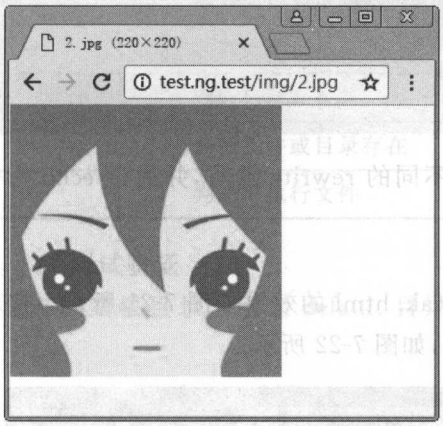


图 7-23 重定向效果

需要注意的是，redirect 和 permanent 在使用时有一定的区别，前者返回的 HTTP 状态码是 302(临时重定向)，使得搜索引擎在抓取新内容的同时保留旧的网址，后者返回的 HTTP 状态码是 301(永久重定向)会让搜索引擎在抓取新内容的同时也将旧的网址永久替换为重定向之后的网址。

7.6 防盗链的配置

盗链是指有一些不良网站，为了在不增加成本的前提下扩充自己站点的内容，直接盗用其他网站的资源链接，而大部分用户又不会发现。这样的做法一方面损害了原网站的合法利益，另一方面又加重了原网站服务器的流量负担。因此，为了保护网站中的资源不被盗链，接下来本节将对如何在 Nginx 服务器中配置防盗链进行详细讲解。

7.6.1 图片防盗链

图片是一个网站中最容易被盗取的资源，如何使用 Nginx 来保护图片不被盗链呢？在讲解之前，首先要了解 HTTP 请求消息中的一个名称为 referer 的字段，它用于保存当前网页的来源 URL 地址。当用户打开一个含有图片内容的网页时，浏览器会在图片的请求消息中将网页的 URL 放在 referer 中，从而使图片所在的服务器能够跟踪到它被显示的网页地址。

因此，若要实现图片防盗链，最简单的防护手段就是判断 referer 的值，来判断当前图片的引用是否合法，一旦检测到来源不是本站，就立即阻止图片的发送，或换成一张禁止防盗链提示的图片。

下面简单讲解 Nginx 中如何进行图片防盗链配置，具体步骤如下。

1. 准备两个网站

在 nginx.conf 文件中，配置两个虚拟主机 www.ng.test 和 www.test.com，用于在网站 www.test.com 中盗用网站 www.ng.test 中的图片链接，配置如下。

```
server {
    listen 80;
    server_name www.ng.test;
    root html/ng.test;
    index  index.html index.htm;
}
server {
    listen 80;
    server_name www.test.com;
    root html/test.com;
    index  index.html index.htm;
}
```

接下来，在网站 www.ng.test 的目录 html/ng.test 中创建 img 子目录，用于存放网页中的图片资源。然后，分别在网站 www.ng.test 和 www.test.com 中创建测试网页，具体如下。

(1) 在网站 www.ng.test 下，创建 index.html 文件，用于展示自己网站中的图片，内容如下。

```
<h1>Welcome to www.ng.test!</h1>
<img src="img/2.jpg" width="200" />
```

(2) 在网站 www.test.com 中盗用网站 www.ng.test 中的图片链接，index.html 的内容如下。

```
<h1>Welcome to www.test.com!</h1>
<img src="http://www.ng.test/img/2.jpg" width="200" />
```

准备完成后，在浏览器中分别访问网站 http://www.ng.test 和 http://www.test.com，如图 7-24 和图 7-25 所示。从图中可以看出 www.test.com 网站从 www.ng.test 中盗链成功。此时按 F12 键查看图片的请求消息，在 Request Headers 中可以查看当前 Referer 的值。

2. 配置图片防盗链

接着，修改网站 www.ng.test 的 server 配置，添加允许文件链出的域名白名单，具体设置如下。

```
location ~* \.(gif|jpg|png|swf|flv)${
    valid_referers  www.ng.test ng.test;
    if($invalid_referer){
        return 403;
```

```
5        }
6    }
```

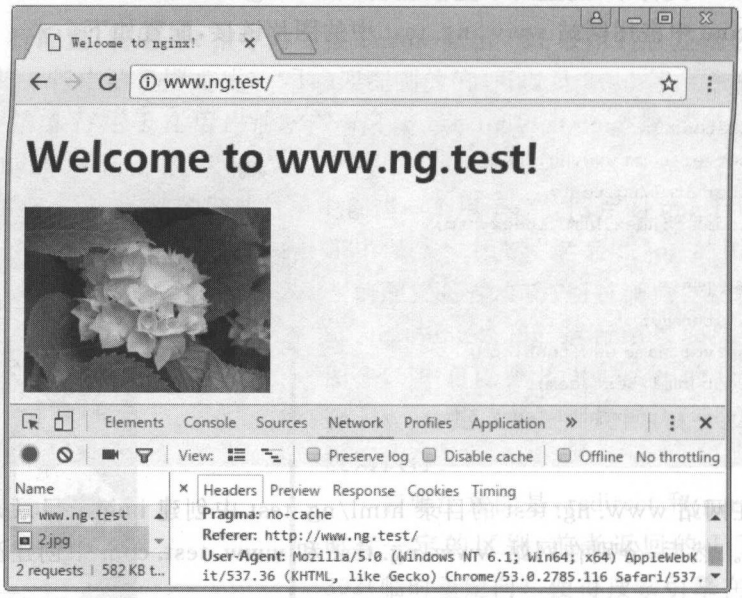


图 7-24 访问网站 www. ng. test

图 7-25 访问网站 www. test. com

上述第 1 行配置，用于匹配文件扩展名为 gif、jpg、png、swf、flv 的资源；第 2 行中的 valid_referers 指令用于设置允许访问资源的网站列表(即白名单)。当请求消息头中的 referer 符合白名单时，内置变量 $invalid_referer 的值为空字符串，否则为 1。因此，通过第 3～5 行的配置，可以禁止白名单之外的网站访问资源，并返回 403 状态码。

值得一提的是，valid_referers 指令的参数可以叠加设置，中间使用空格分隔即可。关于 valid_referers 指令后可以设置的参数值以及相关说明如表 7-6 所示。

表 7-6　valid_referers 可选参数值

参　数　值	说　　明
none	匹配没有 Referer 的 HTTP 请求，如 valid_referers none；
blocked	匹配 HTTP 请求中含有 Referer，但是被防火墙或者代理服务器修改，去掉了 https://或 http://的情况，如 valid_referers blocked；
server_names	允许文件资源链出的域名白名单 ，如 valid_referers www. ng. test ng. test；
string	任意的字符串，如 valid_referers *. ng. test bxg. * ；
regular expression	正则表达式，如 valid_referers ～\.img\.；

3. 验证测试

按照上述配置完成后，访问原网站 http://www. ng. test，可以看到图片。访问盗链网站 http://www. test. com，如图 7-26 所示。从图 7-26 中可以看出，在网站 www. ng. test 设置了图片防盗链后，网站 www. test. com 在盗取时，会返回 403 错误状态码。

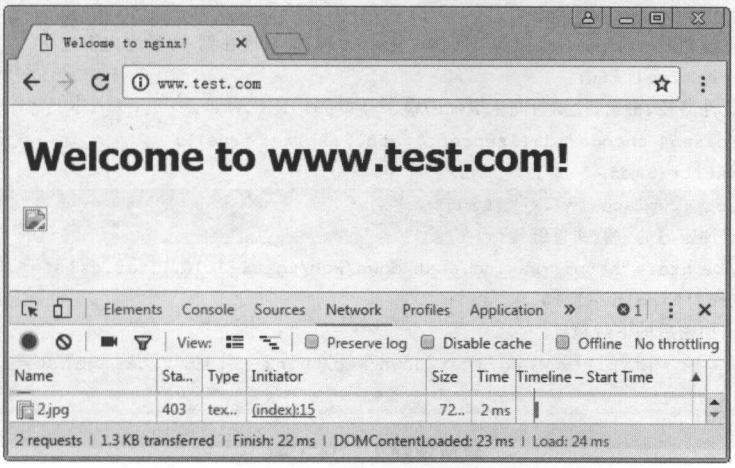


图 7-26　防盗链测试访问

需要注意的是，不是所有的浏览器都会发送 referer 请求头 ，并且 referer 的值还可以被客户端随意修改。也就是说，referer 是可以被伪造的。因此，上述讲解的方式只能用于防范普通用户对图片资源的盗用。

7.6.2　下载防盗链

上一节讲解的图片防盗链只能根据请求消息中的 referer 进行验证，这种方式存在伪造 referer 的问题。而对于图片之外的下载资源，如果不需要考虑资源链接的长期有效性，可以使用 Nginx 提供的 secure_link 和 secure_link_md5 指令来实现下载地址的加密和时效性。

下载防盗链的实现原理就是在服务器端根据特定规则对下载地址进行加密，并在用户请求下载链接时，验证加密链接是否有效，防止用户伪造下载链接。同时，为了避免加密后的链接被盗链，在加密时添加过期时间，使得下载地址只在一定时间内有效，过期后只能到原网站获取新的地址。

接下来，这里以PHP结合Nginx的方式讲解如何实现下载防盗链，具体步骤如下。

(1) 重新编译Nginx

由于指令secure_link和secure_link_md5是由ngx_http_secure_link_module模块提供的，要想使用此模块提供的功能必须在编译Nginx时指定如下参数。

```
--with-http_secure_link_module
```

然后重新生成Nginx的二进制可执行文件，替换掉Nginx安装目录中的可执行文件/usr/local/nginx/sbin/nginx即可。

(2) 创建PHP文件cdown.php，用于生成加密的下载链接，具体代码如下。

```
1  <?php
2  //自定义密钥
3  $secret='ng.test';
4  //下载文件路径
5  $path='/down/web/nginx-1.10.1.tar.gz';
6  //生成过期时间,time()是当前时间,60表示60秒,即从现在到60秒之内不过期
7  $expire=time()+60;
8  //用文件路径、密钥、过期时间生成加密串
9  $md5=base64_encode(md5($secret.$path.$expire, true));
10 $md5=strtr($md5, '+/', '-_');
11 $md5=str_replace('=', '', $md5);
12 //生成加密后的下载文件链接
13 echo '<a href="http://www.ng.test/down/web/nginx-1.10.1.tar.gz?st='.$md5.'&e='.
14 $expire.'">nginx-1.10.1</a>';
15 //输出加密后的下载地址
16 echo '<br>http://www.ng.test/down/web/nginx-1.10.1.tar.gz?st='.$md5.'&e='.
   $expire;
17 ?>
```

上述代码中，第9～11行在生成加密串时，遵循了secure_link_md5指令的加密算法，该算法是根据密钥、URI的MD5哈希值和过期时间组成的，且是由BASE64进行编码的。因此，在PHP中生成的下载链接加密串，必须在使用md5()函数生成哈希值后，再使用base64_encode()函数进行BASE64编码，并对编码后字符串中的+/和=分别使用"-_"和空格进行替换。

第13行代码用于在网页中生成下载链接，同时将加密串和过期时间作为参数st和e添加到下载链接后。第15行代码用于直接输出加密后的下载地址，方便学习查看。

需要注意的是，为了测试成功，需要在域名为www.ng.test的网站根目录下，创建目录down/web/并上传nginx-1.10.1.tar.gz文件，用于用户下载。

(3) 修改Nginx配置文件。

打开Nginx的配置文件，在www.ng.test网站的server块下添加以下配置，验证用户

请求的下载链接是否有效。具体如下。

```
location / {
    secure_link $arg_st,$arg_e;
    secure_link_md5 ng.test$uri$arg_e;
    if($secure_link=""){    #处理加密串不等的情况
        return 403;
    }
    if($secure_link="0"){   #加密串相等时,处理下载链接过期的情况
        return 403;
    }
}
```

上述第 2 行配置中的 secure_link 指令，用于获取从用户端传递的参数，形如“ $arg_参数名称”的内置变量保存了相应的参数值。其中，变量 $arg_st 表示用户传递的加密串，$arg_e 表示过期时间。第 3 行 secure_link_md5 指令后的字符串 ng. test 是服务器指定的加密密钥，该指令用于根据其后的表达式 ng. test $uri $arg_e 生成加密串，并与用户传递过来的 $arg_st 加密串进行对比，对比结果一致则内置变量 $secure_link 的值为 1，否则为空字符串或 0。第 4～9 行配置用于处理匹配不成功时，向客户端返回 403。

需要注意的是，服务器密钥 ng. test 是保护下载链接的唯一凭据，如果密钥泄露，则用户可以自己生成有效的加密下载链接。因此，推荐每隔一段时间更换一次服务器密钥。

(4) 验证测试。

在浏览器中访问 http://www. ng. test/cdown. php，并单击“nginx-1. 10. 1”下载链接进行下载，如图 7-27 所示。

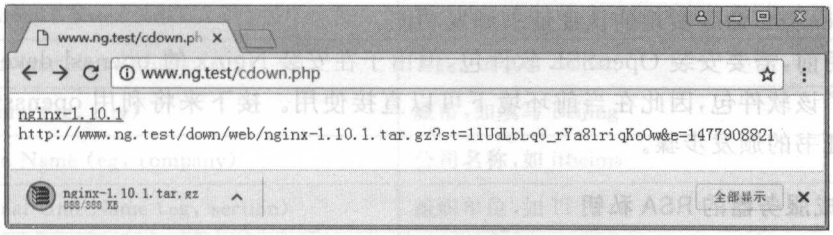


图 7-27　下载防盗链

保持打开后的页面不要刷新，等待超过设定的 1 分钟过期时间后，再下载一次，如图 7-28 所示。从图 7-28 中可以看出，超过了设置的下载过期时间，Nginx 在进行比对时就会失败，返回 403 页面。

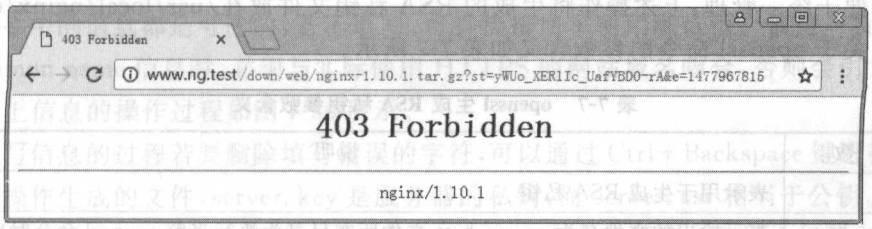


图 7-28　过期的下载链接

7.7 配置HTTPS网站

7.7.1 什么是HTTPS

HTTPS(Hypertext Transfer Protocol Secure)超文本传输安全协议是HTTP(超文本传输协议)、SSL(Secure Sockets Layer)安全套接层和TLS(Transport Layer Security)传输层安全的组合,用于提供加密通信和鉴定网络服务器的身份。随着互联网的飞速发展,网上的支付交易、个人隐私和企业中的敏感信息等越来越受到人们的关注和保护。因此,HTTPS目前已经是所有注重隐私和安全的网站首选。

要想实现HTTPS加密网站,在服务器端首先要获得CA(CertificationAuthority)认证机构颁发的服务器数字证书(CRT),然后浏览器在发起HTTPS请求时会验证服务器的CRT是否合法,若不合法则给出一个warning提示信息;若合法,用户在与网站交互时,所传输的数据都是加密后的数据,达到了安全可靠的效果。

Nginx服务器中的ngx_http_ssl_module模块用于提供HTTPS网站的配置。由于专业的CA机构颁发的证书是收费的,且需要IP地址和域名。在学习阶段,Nginx服务器若要获取数字证书,可以使用OpenSSL开源软件将自己作为CA为自己颁发证书。

7.7.2 颁发认证证书

OpenSSL是一个非常强大的安全套接字层密码库,它包含了主要的密码算法,常用的密钥、证书封装管理以及SSL协议、证书签发等多种功能。其中,OpenSSL程序是由3部分组成的,分别为openssl命令行工具、libcrypto公共加密库和libssl实现SSL协议。这里主要讲解OpenSSL命令实现的认证证书颁发功能。

在使用前,需要安装OpenSSL软件包,但由于在安装Nginx的openssl-devel依赖包时已经安装了该软件包,因此在当前环境下可以直接使用。接下来将利用openssl命令详细讲解认证证书的颁发步骤。

1. 生成服务器的RSA私钥

RSA是HTTPS使用的一种算法,在配置HTTPS前,需要先为服务器生成私钥。

```
[root@localhost ~]#mkdir /usr/local/nginx/conf/ssl
[root@localhost ~]#cd /usr/local/nginx/conf/ssl
[root@localhost ssl]#openssl genrsa -out server.key 2048
```

为了便于统一管理,上述操作将生成的RSA私钥文件放在/usr/local/nginx/conf/ssl/目录中。关于openssl命令的参数含义如表7-7所示。

表7-7 openssl生成RSA私钥参数含义

参　数	说明
genrsa	表示用于生成RSA私钥
out server.key	表示输出的文件名为server.key,文件所在目录为执行当前openssl命令时所在目录
2048	密钥长度为2048(推荐至少2048,长度越长,安全性越强)

在执行完上述 openssl 命令设置后，通过 ls 查看生成的 server. key，如图 7-29 所示。

```
[root@localhost ssl]# openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................+++
.....................................+++
e is 65537 (0x10001)
[root@localhost ssl]# ls
server.key
[root@localhost ssl]# head -3 server.key
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA5lNsO4d/0obNAQ7fe8pfEvrjffpjc1CyU3NiUHfrv8I5fhTz
l6o4VKiqQhc8YtPpY+iigJmdGvVKGwI5KzTTAKRsbqb871yQqH7D3QX9Vlz901ZK
[root@localhost ssl]#
```

图 7-29　生成服务器的 RSA 私钥

2. 生成服务器的 CSR 证书请求文件

CSR 证书请求文件是服务器的公钥，用于提交给 CA 机构进行签名。生成 CSR 的命令如下。

```
[root@localhost ssl]#openssl req -new -key server.key -out server.csr
```

在上述命令中，req 表示证书签发申请，-new 表示新请求，-key server. key 指定私钥为 server. key，out server. csr 表示生成的 CSR 证书请求文件的名称为 server. csr。

在执行上述命令的过程中，程序会要求用户填写一些信息，具体如表 7-8 所示。

表 7-8　openssl 生成 CSR 参数含义

参　数	说　明
Country Name (2 letter code)	符合 ISO 的 2 个字母的国家代码，如中国 CN
State or Province Name (full name)	省份，如填写 Beijing
Locality Name (eg, city)	城市，如填写 Beijing
Organization Name (eg, company)	公司名称，如 itheima
Organizational Unit Name (eg, section)	组织单位，如 IT
Common Name (eg, your websites domain name)	使用 SSL 加密的网站域名，如 www. test. com
Email Address	邮件地址，可以省略
A challenge password	有些 CA 机构需要此密码，通常省略即可
An optional company name	可选的公司名称，可以省略

表 7-8 中的信息都是可选的，若不填写，可直接按回车使用默认值。值得一提的是，在填写 common name 信息时，必须与实际使用 HTTPS 的网站域名吻合，否则会引发浏览器警报。以上信息的操作过程如图 7-30 所示。

在填写信息的过程若要删除填写错误的字符，可以通过 Ctrl＋Backspace 键进行操作。

以上操作生成的文件，server. key 是服务器的私钥，而 server. csr 相当于公钥。利用公钥可以对数据进行加密，加密后只有用私钥才能解密。而私钥用于对数据进行数字签名，签名后的数据可以利用公钥进行验证。因此，用户应妥善保管私钥，一旦泄露或丢失将无法保

证安全。

```
[root@localhost ssl]# openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:CN
State or Province Name (full name) []:Beijing
Locality Name (eg, city) [Default City]:Beijing
Organization Name (eg, company) [Default Company Ltd]:itheima
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:www.test.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@localhost ssl]#
```

图 7-30 生成服务器的CSR证书请求文件

3. CA为服务器认证证书

```
[root@localhost ssl]#openssl x509 -req -days 30 \
-in server.csr -signkey server.key -out server.crt
```

上述指令用于使用CA的私钥server.key为服务器的CSR证书申请文件server.csr进行签名认证。其中，x509是自签名证书格式，days 30用于设置签发证书的有效期为30天。如图7-31所示。

```
[root@localhost ssl]# openssl x509 -req -days 30 \
> -in server.csr -signkey server.key -out server.crt
Signature ok
subject=/C=CN/ST=Beijing/L=Beijing/O=itheima/OU=IT/CN=www.test.com
Getting Private key
[root@localhost ssl]#
```

图 7-31 CA为服务器认证证书

在CA利用私钥签名证书后，该证书将用于浏览器验证请求的网站是否真实，防止网络通信过程中被伪造。浏览器保存了受信任的CA机构的公钥，在请求HTTPS网站时，会利用CA公钥验证服务器的证书，并检查域名是否吻合、证书是否过期、证书是否已经被吊销等。由于当前的证书是服务器自己作为CA签名的，因此浏览器无法信任，会出现警告，但不影响学习和测试。

CA的认证证书颁发完成后，在Nginx中添加CRT证书和服务器的私钥即可完成HTTPS网站的配置。在加密通信时，浏览器通过网站的证书可以获得服务器的公钥，然后利用公钥加密请求信息，Nginx收到后再利用服务器私钥解开信息。

浏览器在证书认证后，会在请求信息中包含一个自动生成的高强度密钥(或称为随机数)，服务器收到后会利用该密钥加密响应信息。由于证书认证和RSA非对称加密的过程复杂，为了提高效率，在证书认证后的一段时间内是直接利用这个密钥进行对称加密通信的。

7.7.3　配置 HTTPS 网站

在 Nginx 服务器中配置 SSL 服务，首先需要在编译安装 Nginx 时添加对 ngx_http_ssl_module 模块的支持，在 3.2.2 节中讲解编译安装 Nginx 时已经添加了对该模块的支持。接下来，打开 Nginx 的配置文件 nginx.conf，在该配置文件中配置一个支持 HTTPS 的网站 www.test.com，常用设置如下所示。

```
1   server {
2       listen 443;
3       server_name www.test.com;
4       root html/test.com;
5       ssl on;
6       ssl_certificate      /usr/local/nginx/conf/ssl/server.crt;
7       ssl_certificate_key /usr/local/nginx/conf/ssl/server.key;
8   }
```

上述第 2 行指令，用于设置提供 HTTPS 服务的端口号，443 端口专门用于 HTTPS。第 5 行指令用于开启 Nginx 对 SSL 的支持，第 6 行用于指定 CA 认证后的 CRT 文件的路径，第 7 行用于指定服务器私钥的路径。

按照上述设置修改完成后，平滑重启 Nginx 使配置生效。接下来，在浏览器中访问 https://www.test.com，出现如图 7-32 所示的效果。这是由于自行颁发的 SSL 证书虽然能够实现加密传输功能，但浏览器并不信任，因此会出现提示信息。

图 7-32　访问 HTTPS 网站

接着，选择“高级”项，选择“继续前往 www.test.com(不安全)”后，即可访问该网站下的文件，如图 7-33 所示。

通过 F12 键打开开发者工具，选择 View certificate 可以查看证书，如图 7-34 所示。

从图 7-34 中可以看到，该证书的颁发者、使用者和有效日期，接着可以切换到“详细信息”选项卡，查看具体的配置；切换到“证书路径”选项卡，可以查看该证书的路径及状态。

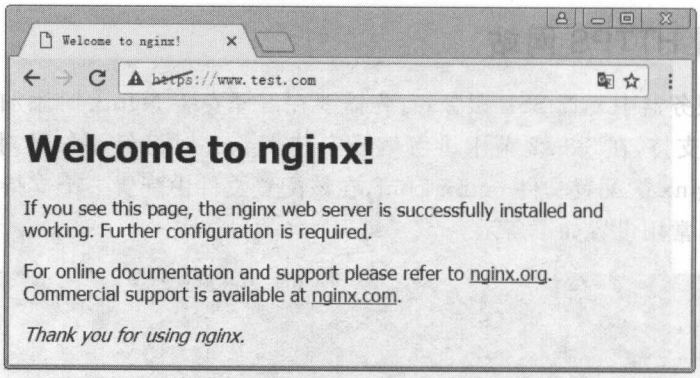


图 7-33 安全访问

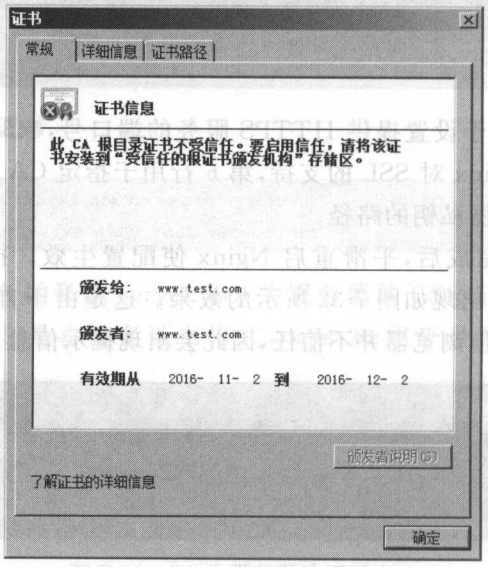


图 7-34 查看证书

本章小结

通过本章的学习，希望读者能够了解 Nginx 模块化设计的好处、作用以及模块的分类；学会查看 Nginx 手册、利用 echo 进行调试输出、学会查看网站的响应状态以及替换响应内容；要求熟练掌握 gzip 的网页压缩技术、rewrite 重写与重定向的实现、图片防盗链、下载防盗链的配置，以及能够理解 HTTPS 服务实现的原理和具体配置。

课后练习

一、填空题

1. Nginx 中_____命令可以用于指定 MIME 类型。

2. 内置变量_____获取从 HTTP 请求开始到当前时间的秒数。

二、判断题

1. 在 http 块中，可以使用 stub_status 命令查看所有网站的状态。（　　）
2. echo 指令的-n 参数用于设置输出内容后不换行。（　　）
3. 默认情况下，Nginx 只安装核心模块，其他模块可根据实际需求安装。（　　）

三、选择题

1. 下列选项中，作为 rewrite 指令的参数可以实现永久重定向的是（　　）。
   A. last　　B. redirect　　C. permanent　　D. break
2. echo 指令调试输出的内容中若包含“-”，可以使用（　　）进行转义。
   A. /　　B. \　　C. -　　D. --
3. 下列选项中，可以在./configure 中添加 echo-nginx-module 模块的选项是（　　）。
   A. --prefix　　B. --with-echo-nginx-module
   C. --add-module　　D. --sbin-path
4. 下面关于压缩级别的描述错误的是（　　）。
   A. 压缩级别越高，则压缩的时间越长
   B. 压缩率级别过高会造成 Nginx 能同时处理请求的响应能力下降
   C. 压缩级别越低，则 CPU 消耗越多，所需时间越长
   D. 压缩级别越高，则 CPU 消耗越大

四、操作题

Nginx 中除了常用的 http 模块外，还有 stream 模块，它用于一般的 TCP 代理和负载均衡，使用方式与 http 模块基本相同。下面请在编译安装 Nginx 时激活 steam 模块，并利用 stream 模块完成服务器的负载均衡。

第 8 章

高可用负载均衡集群

学习目标

- 掌握 Nginx 的配置优化；
- 掌握 LNMP 分布式集群的搭建；
- 掌握 Nginx＋Keepalived 高可用方案的部署。

经过前面的学习，相信读者已经掌握 Nginx 的常用配置和操作。为了提高综合运用能力，本章将会针对高并发、高负载网站需求的具体实现进行讲解，包括 Nginx 的配置优化、LNMP 分布式集群，以及 Nginx＋Keeplived 高可用方案，利用这些技术来增强网站的性能和可靠性。

8.1 Nginx 配置优化

软件的配置优化，是指在硬件资源允许的前提下，将硬件资源集中到特定的工作中，其本质是减少为其他工作保留的余力，来换取在指定工作上的性能提升。Nginx 的默认设置是为了平衡各种工作场景而准备的通用方案，在特定工作中并不能发挥最好的性能。下面将对 Nginx 的配置优化进行讲解。

8.1.1 连接数优化

当 Nginx 作为负载均衡服务器时，能够接收的并发连接应该越多越好。为了测试服务器的并发能力，可以利用 Apache 中提供的 ApacheBench 工具，该工具可以在一台计算机中向一个 URL 地址发送大量的并发请求，然后检测服务器响应这些请求花费了多少时间，有多少请求处理失败。利用 ApacheBench 工具可以模拟一台服务器被大量客户端并发连接的情况，以测试服务器的并发能力。

为了准备测试环境，使用 VMware 部署两台虚拟机，如图 8-1 所示。

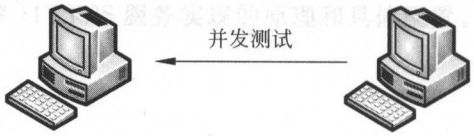


图 8-1 测试环境

在图 8-1 中，服务器是待测试机，安装了 Nginx 并使用默认配置；客户端使用 ApacheBench 工具进行并发访问测试。

1. 使用 ApacheBench 工具

在客户端中切换到 Apache 安装目录的 bin 目录，该目录下的 ab 程序就是 ApacheBench 工具。下面通过一个示例来演示 ApacheBench 工具的使用，具体如下。

```
[root@localhost ~]#cd /usr/local/apache2/bin
[root@localhost bin]#./ab -n10 -c2 http://192.168.78.3/
```

上述操作中，ab 命令的选项-n 表示发送的请求总数，-c 表示并发数，后面的网址是请求的服务器 URL 地址。ApacheBench 目前只能使用 HTTP 1.0 协议进行请求。

为了使读者更好地理解 ApacheBench 的测试报告，下面通过一个具体的示例进行演示。

```
[root@localhost bin]#./ab -n500 -c500 http://192.168.78.3/
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
⋮
Finished 500 requests                                    #完成 500 个请求
Server Software:          nginx/1.10.1                   #服务器软件
Server Hostname:          192.168.78.3                   #服务器主机名
Server Port:              80                             #服务器端口号
Document Path:            /                              #文档路径
Document Length:          612 bytes                      #文档大小
Concurrency Level:        500                            #并发数
Time taken for tests:     0.121 seconds                  #完成测试所用的时间
Complete requests:        500                            #完成的请求数
Failed requests:          0                              #失败的请求数
Total transferred:        422500 bytes                   #总传输数据量大小
HTML transferred:         306000 bytes                   #HTML 数据量大小
Requests per second:      4119.29 [#/sec] (mean)         #平均每秒请求数
Time per request:         121.380 [ms] (mean)            #平均每次并发请求所用时间
Time per request:         0.243 [ms] (mean, ...)         #平均每个请求所用时间
Transfer rate:            3399.22 [Kbytes/sec] received  #平均每秒传输的数据量
⋮
Percentage of the requests served within a certain time (ms)   #各请求的时间分布如下
  50%     57                                             #50%的请求花费了 57 毫秒
  ⋮
  99%     68                                             #99%的请求花费了 68 毫秒
 100%     68 (longest request)                           #所有请求花费了 68 毫秒
```

从上述测试报告可以看出，服务器能够承受 500 个并发连接，总共耗时约 0.12 秒。接下来将请求总数和并发数设置为 1500，会发现 ApacheBench 无法进行测试，出现如下提示。

```
socket: Too many open files (24)
```

该提示表示打开的文件超出了系统限制。按照 Linux 一切皆文件的理念，连接数也是文件，Linux 系统默认限制了一个进程最多打开的文件数量。通过 ulimit -a 可以查看当前

系统的限制，具体如下。

```
[root@localhost ~]#ulimit -a | grep open
open files                      (-n) 1024
```

从上述结果中可以看出，当前对于文件打开数量(open files)的限制为1024，使用ulimit -n命令可以临时更改这个数量。若要每次开机后自动修改，可以将命令写入/etc/profile文件中。下面修改打开文件数量为65 535，然后重新运行ab命令进行测试。

```
[root@localhost bin]#ulimit -n 65535
[root@localhost bin]#./ab -n1500 -c1500 http://192.168.78.3/
```

上述操作只更改了客户端Linux系统，而服务器端仍然存在限制，因此在1500并发连接情况下，会出现失败的请求，在ApacheBench的测试报告中会看到如下的信息。

```
Complete requests:  1500                    #完成发送的请求数
Failed requests:    859                     #失败的请求数
```

从信息中可以看出，在1500的并发数下Nginx服务器出现了失败的情况，此时若查看Nginx的日志文件error.log，可以看到Too many open files的错误记录。

2. 优化Nginx连接数

为了使Nginx能够承载更高的并发数，可以编辑conf/nginx.conf配置文件进行配置，具体如下。

```
worker_processes  auto;
worker_rlimit_nofile 65535;
events {
    worker_connections 65535;
    multi_accept on;
}
```

在上述配置中，worker_processes指令用于指定工作进程的个数，设置为auto时Nginx将根据CPU的核心数来控制；worker_rlimit_nofile用于设置最多打开的文件数量；worker_connections用于设置每个工作进程可接收的连接数；multi_accept表示是否允许一个工作进程响应多个请求。

Nginx支持select、poll、kqueue、epoll等多种类型的连接处理方式，在默认情况下会自动选择最适合系统的方式。将错误日志级别设置为info时，可以查看当前Nginx使用的方式，如下所示。

```
[root@localhost conf]#echo 'error_log logs/error.log info;' >>nginx.conf
[root@localhost conf]#rm -f ../logs/error.log
[root@localhost conf]#nginx -s reload
[root@localhost conf]#cat ../logs/error.log | head -2
2016/10/27 17:46:43 [notice] 5024#0: signal process started
2016/10/27 17:46:43 [notice] 4928#0: using the "epoll" event method
```

从上述操作可以看出，Nginx 在当前系统中使用了 epoll 事件方式，epoll 是 Linux 平台上性能非常高的一种多路 I/O 复用模型。在系统内核和 glibc(C 运行库)支持的情况下，Nginx 会自动使用 epoll 方式。

接下来，使用 ApacheBench 工具进行 2000 并发连接数测试，具体操作如下。

```
[root@localhost bin]#./ab -n2000 -c2000 http://192.168.78.3/
⋮
Concurrency Level:      2000
Time taken for tests:   0.379 seconds
Complete requests:      2000
Failed requests:        0
```

从测试报告中可以看出，Nginx 完成了并发测试，没有出现失败的请求。

8.1.2　客户端请求限制

在完成 Nginx 连接数优化后，有一个问题值得注意，就是在真实上线环境中，如果服务器遇到同一个 IP 地址发送了 2000 个并发请求，很可能是遇到网络攻击，如果没有任何防御措施，会消耗服务器大量的资源。企业一般会通过部署专业的防火墙设备来阻挡攻击，而 Nginx 本身也具有这样的功能，下面将介绍如何利用 Nginx 来对客户端的请求进行限制。

1. 限制同一个 IP 的并发数

通过 limit_conn 指令可以限制并发连接数，在 conf/nginx.conf 配置文件中进行如下配置即可。

```
http {
    limit_conn_zone $binary_remote_addr zone=perip:10m;
    limit_conn perip 10;
    ⋮
}
```

在上述配置中，limit_conn_zone 指令用于开辟一个共享内存空间保存客户端 IP，空间名称为 perip，空间大小为 10MB；limit_conn 指令用于限制连接数量；预定义变量 $binary_remote_addr 保存了用二进制表示的当前客户端 IP 地址。上述配置生效后，Nginx 将对于同一个 IP 地址只允许 10 个并发连接，当超过时返回 503(服务暂时不可用)错误。另外，limit_conn 指令也可以在 server 和 location 块中使用，用于实现不同级别的控制。

在客户端使用 ApacheBench 工具进行并发测试，可以看到 100 个请求中只有 10 个是正常的。

```
[root@localhost bin]#./ab -n100 -c100 http://192.168.78.3/
⋮
Complete requests:      100
Failed requests:        90  (Connect: 0, Receive: 0, Length: 90, Exceptions: 0)
Non-2xx responses:      90  #有 90 个响应的状态码不是 2xx
```

2. 限制虚拟主机的并发数

在使用 limit_conn_zone 指令时，也可以用共享内存空间保存虚拟主机名（$server_name），实现对虚拟主机的并发数进行限制，具体配置如下。

```
http {
    ⋮
    limit_conn_zone $server_name zone=perserver:10m;
    server {
        listen 80;
        server_name localhost;
        limit_conn perserver 20;
        ⋮
    }
}
```

为了测试虚拟主机并发限制是否有效，将前面的 IP 并发限制取消后，在客户端使用 ApacheBench 工具进行并发测试，可以看到 100 个请求中只有 20 个是正常的。

```
[root@localhost bin]# ./ab -n100 -c100 http://127.0.0.1/
⋮
Complete requests:      100
Failed requests:        80  (Connect: 0, Receive: 0, Length: 80, Exceptions: 0)
Non-2xx responses:      80  #有 80 个响应的状态码不是 2xx
```

由于上述配置只针对 localhost 主机，如果测试其他虚拟主机，则没有并发限制。

3. 限制响应的传输速率

Nginx 的 limit_rate 指令用于限制服务器在响应时传输数据到客户端的速率，可以在 http、server、location、location 中的 if 块中使用。下面在 http 块中进行演示，具体如下。

```
http {
    limit_rate 100k;
    limit_rate_after 10m;
    ⋮
}
```

在上述配置中，limit_rate 用于限制每个连接的传输速率（每秒 100KB）；limit_rate_after 用于在已经传输指定大小的数据后再进行限速，从而实现只针对大文件限制下载速度。如果省略 limit_rate_after 指令，则无论文件大小是多少，都会进行限速。

为了测试限速是否生效，在站点文档目录下创建一个 100MB 的测试文件，具体操作如下。

```
[root@localhost ~]# cd /usr/local/nginx/html
[root@localhost html]# dd if=/dev/zero of=100mb.test bs=1M count=100
```

上述命令中，dd 用于按照指定大小的块复制文件，if 表示输入文件，of 表示输出文件，

bs 表示每个块的大小，count 表示块的数量。输入文件/dev/zero 是一个可以无限读取出空字符(即 ASCII 中的 0)的虚拟文件，读取后会保存到当前目录下的 100mb. test 文件中。

在客户端通过 wget 命令下载文件，测试下载速度，测试结果如下。

```
[root@localhost bin]#wget http://192.168.78.3/100mb.test
⋮
Length: 104857600(100M)[application/octet-stream]
Saving to: "100mb.test"
 12%[======>                              ] 12,945,600   100KB/s  eta 16m 46s
```

从结果中可以看出，在客户端下载了超过 10MB 的数据后，Nginx 成功将下载速度限制为 100KB/s。由于 wget 显示的下载速度是平均值，所以会看到速度减慢的过程，但实际上并不是匀速下降的。

8.1.3　浏览器缓存优化

服务器可以通过响应消息控制浏览器缓存，和缓存相关的响应头有 Expires、Cache-Control、ETag、Last-Modified 等。其中，Last-Modified 和 ETag 由 Nginx 自动生成，前者表示最后修改的时间，后者用于浏览器判断内容有无改变；Cache-Control 比较复杂，通常根据实际需要进行控制；Expires 表示该资源的过期时间，如果没有过期则浏览器不会发起 HTTP 请求。

在优化浏览器缓存时，通常会对静态资源(如图片、CSS 文件、JavaScript 文件)进行优化。一个内容丰富的网页中会包含大量的静态资源，在没有浏览器缓存的情况下，每个资源都要请求服务器。由于网页中的静态资源是不经常发生变化的，服务器可以控制静态资源的过期时间，使浏览器对于未过期的资源直接从缓存中读取，以减少请求次数，使网页整体加载速度更快。

接下来在 Nginx 配置文件中通过 expires 指令为静态资源设置过期时间，将图片、swf(Flash 动画)文件设置为 30 天后过期，将 css、js 文件设置为 12 小时后过期，具体配置如下。

```
server {
    ⋮
    location ~\.(gif|jpg|jpeg|png|bmp|swf)${
        expires      30d;
    }
    location ~\.(css|js)${
        expires      12h;
    }
}
```

为了测试缓存控制是否生效，在站点文档目录中创建一个网页 test. html，编写代码如下。

```
<link rel="stylesheet" type="text/css" href="style.css" />
<img src="photo.png" />
```

上述代码在网页中引入了 style. css 和 photo. png，读者可以自行准备这两个文件。打

开浏览器进行访问测试，通过开发者工具可以查看资源的过期时间，如图 8-2 所示。

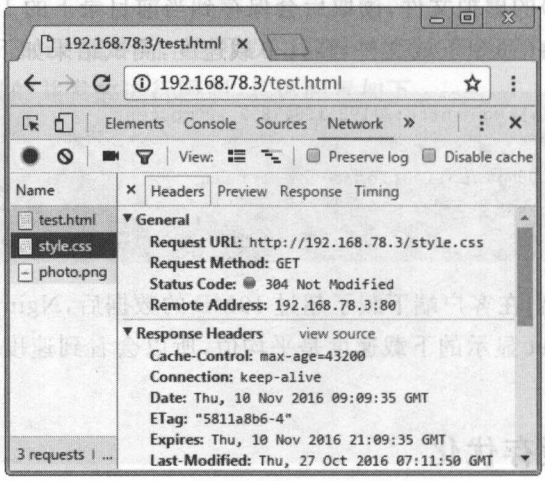


图 8-2 查看缓存过期时间

如果此时执行刷新操作，则浏览器会请求所有的资源，而若在地址栏中单击 URL 并按 Enter 键(重新打开网页)，则会看到静态资源显示为 from cache(来自缓存)，如图 8-3 所示。

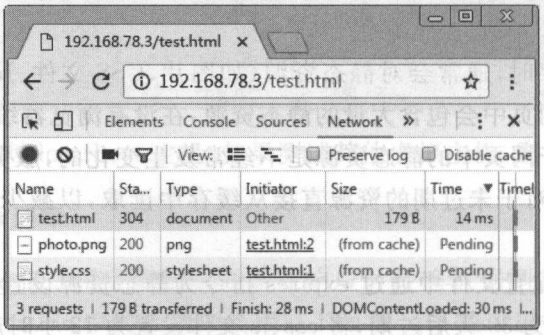


图 8-3 查看缓存情况

值得一提的是，即使服务器没有设置 Expires，浏览器也会基于常见的静态资源扩展名自动缓存；但若设置 Expires，可以使缓存具有更长的有效期。另外，当服务器需要更新静态资源时，可以修改 HTML 中引入的地址，利用 URL 参数让浏览器重新请求静态资源，如改为 style. css? ver=1. 2。

8.2 LNMP 分布式集群

8.2.1 什么是集群

集群(cluster)是指将多台服务器集中起来一起进行同一种服务。相比一台服务器，集群的优势在于将负载均衡到每台服务器上，从而承载更多的工作量。而且集群具有很强的可靠性，当其中一台服务器发生故障时，不会造成整个服务中断。集群还具有成本上的优势，这是因为当一台计算机的计算能力达到一定程度时，就会产生瓶颈，即付出巨大的成本

后只换来少量的性能提升。如果利用集群架构，可以专门购置一批高性价比的服务器，用较少的成本就能得到可观的性能提升。

目前许多互联网公司都利用集群技术增强服务的计算能力和可靠性，Nginx 的反向代理和负载均衡技术就可以应用在 Web 服务器集群中，图 8-4 演示了一个基于 Nginx 的负载均衡集群架构。

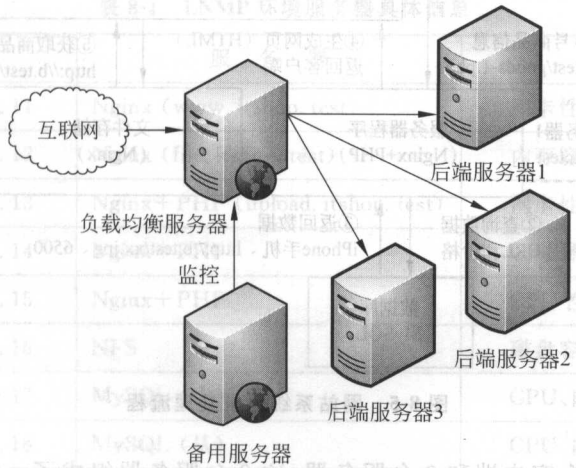


图 8-4　负载均衡集群

在图 8-4 所示的集群中，只有一台负载均衡服务器接入互联网，这台服务器的主要工作是承担网络吞吐压力，而与业务有关的计算工作分摊到 3 台后端服务器中。一台服务器能够承载的并发量和业务的计算量有关，计算量越大则并发量越少。由于负载均衡服务器不承担计算任务，因此可接受极高的并发量，而单台后端服务器能承受的并发量少，但是可以通过增加数量来扩大整体的并发量。

由于负载均衡服务器只有一台，一旦发生故障则整个集群无法工作，为此可以增加备用服务器来监控当前工作的服务器是否正常，一旦发生故障时自动代替原服务器，从而达到高可用的效果。

8.2.2　LNMP 分布式部署

分布式和集群都是为提高服务器处理能力而设计的，其区别是集群由多台服务器共同完成一件工作，而分布式是将工作进行业务拆分，然后由多种不同的服务器进行处理。集群是一种串行的工作方式，虽然服务器数量多，但是对客户端而言只有其中一台服务器处理了请求；而分布式是一种并行的工作方式，由于业务是拆分的，客户端需要向多台服务器发送请求，每个服务器各司其责才能完成任务。

从上述描述可以看出，分布式和集群各有优缺点，但对于规模庞大的网站来说，可以选取两者的优点，将业务拆分到多个集群中，形成分布式集群架构。

一个典型的网站系统主要由程序（如 Java、PHP）、数据库（如 Oracle、MySQL）和文件存储系统（保存图片等数据）组成。其中，程序主要用于动态生成网页、处理用户的输入输出和一些计算工作；数据库主要用于存储和管理网站的数据；文件存储系统主要用于提供文件上

传和下载服务。图 8-5 演示了网站系统对于用户请求的处理流程。

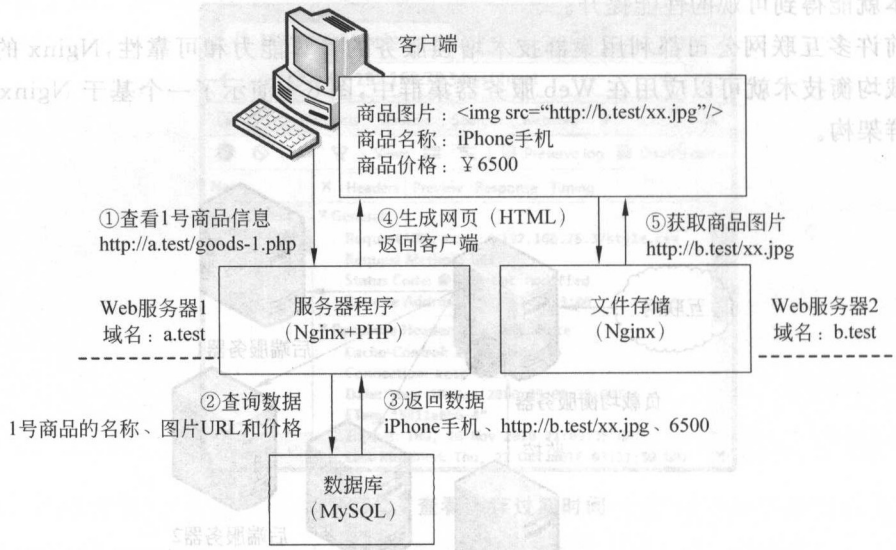


图 8-5　网站系统请求处理流程

图 8-5 中共有 1 个客户端和 3 台服务器，这 3 台服务器组成了一个购物网站。假设网站开发人员在 Web 服务器 a. test 中编写一个 goods-1. php 脚本文件，用于显示 1 号商品的信息。当客户端请求这个文件时，服务器端程序就会按照脚本执行，到数据库中查询出商品信息数据，将数据自动填入开发人员写好的 HTML 模板中，形成一个有内容的网页返回给客户端。由于网站需要大量的图片，因此通过 Web 服务器 b. test 专门负责文件的存储和访问。

从上述流程可以看出，相比 1 台服务器同时负责程序、数据库和文件存储的情况，拆分成 3 台服务器以后，整体服务能力将会有明显提升，而且还可以继续扩展。下面介绍如何搭建一个基于 LNMP 分布式集群的网站系统，其主要结构如图 8-6 所示。

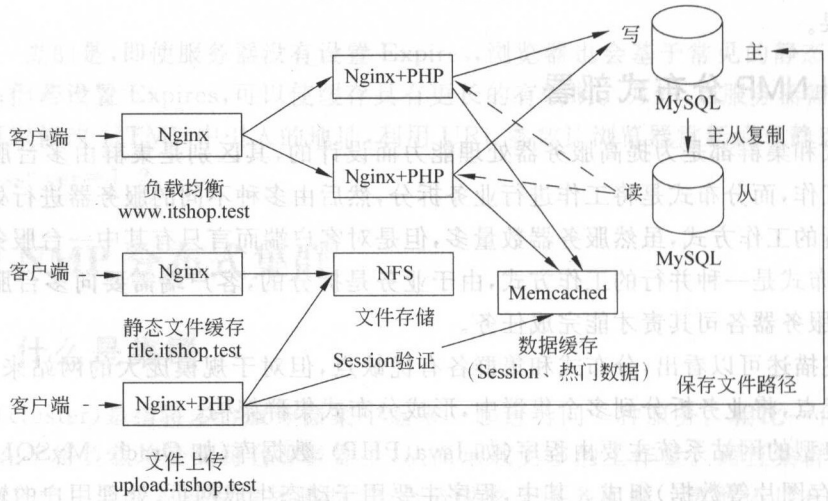


图 8-6　LNMP 分布式部署

在图 8-6 中，总共有 9 台服务器，程序部分由 1 台负载均衡和 2 台 Nginx＋PHP 服务器组成，数据库部分由 2 台 MySQL 服务器组成，文件存储部分由 2 台 Nginx（其中一台是 Nginx＋PHP）和 1 台 NFS 服务器组成。还有 1 台 Memcached 服务器作为数据缓存，用于处理读写频繁的热门数据。

为了便于读者在本地搭建测试环境，下面通过表 8-1 列举每台服务器的具体信息。

表 8-1　LNMP 环境服务器具体信息

编号	IP	服　务　器	硬件侧重点
1	192.168.78.11	Nginx（www.itshop.test）	网卡性能
2	192.168.78.12	Nginx（file.itshop.test）	内存容量、磁盘性能
3	192.168.78.13	Nginx＋PHP（upload.itshop.test）	网卡性能
4	192.168.78.14	Nginx＋PHP	CPU 性能
5	192.168.78.15	Nginx＋PHP	CPU 性能
6	192.168.78.16	NFS	磁盘容量
7	192.168.78.17	MySQL（主）	CPU、内存、磁盘整体性能
8	192.168.78.18	MySQL（从）	CPU、内存、磁盘整体性能
9	192.168.78.19	Memcached	内存容量

在表 8-1 中，编号为 1～5 的服务器的搭建步骤在本节进行讲解，其他服务器如何搭建将会在后面的小节中讲解。

1. 部署 Linux 服务器

1）安装 CentOS

创建 1 号服务器，按照前面讲解的步骤安装 CentOS6.8 系统，选择最小化安装方式即可。安装完成后配置网络，具体命令如下。

```
[root@localhost ~]#vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

打开配置文件后，参考表 8-2 进行编辑，从而使虚拟机能够访问网络。

表 8-2　网卡配置

ONBOOT＝yes	BOOTPROTO＝static	IPADDR＝192.168.78.11
NETMASK＝255.255.255.0	GATEWAY＝192.168.78.2	DNS1＝192.168.78.2

修改完成后保存网卡配置文件，执行 service network reload 命令重新加载配置。在配置生效后，使用 Xshell 远程终端连接服务器进行操作。

在完成 1 号服务器的 Linux 系统安装后，可以利用 VMware 克隆出其他服务器。按照如图 8-7 所示的流程进行虚拟机克隆可简化许多重复的操作。

在图 8-7 中，6、7、9 号服务器不需要安装 Nginx，因此可在 1 号服务器安装 Nginx 之前进行克隆。然后在 1 号服务器安装 Nginx 后，再克隆出 2、3 号服务器。需要注意的是，虚拟

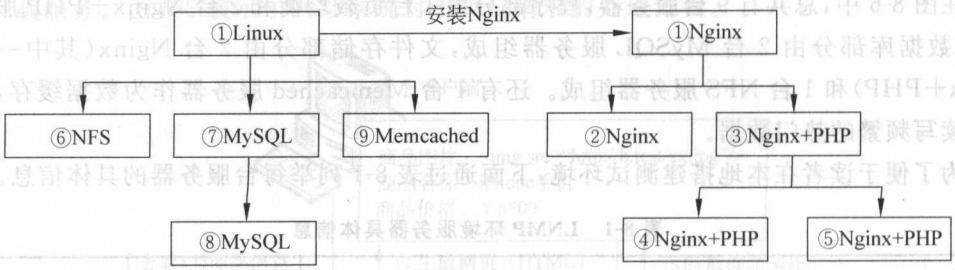


图 8-7 虚拟机克隆流程

机的克隆和 Nginx 的部署将会在后面的步骤中进行操作，此处读者仅了解整体流程即可。

2）内存配额控制

由于实验环境需要 9 台服务器同时运行，因此应控制好每个虚拟机的内存配额，防止物理机内存不足。若物理机有 8GB 内存，推荐为每台虚拟机分配 512MB；若物理机有 4GB 内存，则可以在虚拟机安装 CentOS 系统后利用 VMware 降低内存大小为 256MB；若物理机内存低于 4GB，则建议减少虚拟机的数量，在一台虚拟机中安装多个服务，利用不同的端口号来区分。

3）编写网卡配置脚本

由于每台虚拟机需要更改 MAC 地址和 IP 地址，我们可以编写 shell 脚本来简化这些操作。执行“vi netconfig. sh”命令创建脚本文件，编写代码如下。

```
1  #!/bin/bash
2  eth0=/etc/sysconfig/network-scripts/ifcfg-eth0
3  mac=`ifconfig -a | grep -o HWaddr.*| cut -c 8-24`
4  if [ "$1" = "" ] || [ "$mac" = "" ]; then exit 3; fi
5  sed -i 's/IPADDR=.*/IPADDR=192.168.78.1'$1'/g' $eth0
6  sed -i 's/HWADDR=.*/HWADDR='$mac'/g' $eth0
7  sed -i 's/UUID=.*/UUID='`uuidgen`'/g' $eth0
8  start_udev
```

上述代码中，第 3 行用于获取当前网卡的 MAC 地址；第 5～7 行用于对 eth0 网卡配置文件中的 IP 地址、MAC 地址和 UUID（唯一识别码）执行文本替换。其中，IP 地址的末尾数字通过参数传入，MAC 地址替换为新 MAC 地址，UUID 通过 uuidgen 命令自动生成。最后一行的 start_dev 命令用于更新设备，更新后 eth0 网卡将恢复可用。另外，为了防止脚本意外执行导致修改出错，第 4 行代码判断了 $1 或 $mac 是否为空，如果为空则退出脚本。

保存文件后，执行 chmod +x netconfig. sh 命令为脚本添加可执行权限。

4）克隆虚拟机并配置网卡

使用 VMware 基于 1 号服务器克隆出 3 台新虚拟机，分别作为 6、7、9 号服务器。在克隆后启动新虚拟机进行网卡配置，以 6 号服务器为例，执行如下命令进行配置。

```
#① 执行脚本，传入参数 6(表示更改 IP 地址为 192.168.78.16)
[root@localhost ~]#./netconfig.sh 6
```

```
Starting udev:      [ OK ]
#② 查看更改后的网卡信息
[root@localhost ~]#ifconfig
eth0        Link encap:Ethernet        HWaddr 00:0C:29:8F:15:2B
            inet addr:192.168.78.16    Bcast:192.168.78.255  Mask:255.255.255.0
:
```

从运行结果可以看出，执行脚本后，网卡的 MAC 地址和 IP 地址已经修改成功。按照同样的方式，再为 7、9 号服务器进行网卡配置即可。

2. 部署 Nginx 环境

完成操作系统的安装后，开始部署 Nginx 环境。编号为 1 的负载均衡服务器用于将工作分摊给 4、5 号服务器；2 号服务器用于缓存经常访问的文件以减轻后端服务器的压力，具体配置将会在后面搭建 NFS 文件服务器时再进行讲解。下面开始安装 Nginx 并进行虚拟机克隆。

1) 安装 Nginx

在 1 号服务器中安装 Nginx，具体操作步骤如下。

```
#① 安装依赖包
[root@localhost ~]#yum -y install gcc pcre-devel openssl-devel
#② 解压文件
[root@localhost ~]#tar -zxvf nginx-1.10.1.tar.gz
[root@localhost ~]#cd nginx-1.10.1
#③ 编译安装 Nginx,增加 http_realip_module 模块(后面会用到)
[root@localhost nginx-1.10.1]#./configure --with-http_ssl_module \
--with-http_realip_module && make && make install
#④ 添加环境变量、创建服务脚本、设置开机启动
[root@localhost nginx-1.10.1]#cd /usr/local/nginx/sbin
[root@localhost sbin]#ln -s `pwd`/nginx /usr/local/sbin/nginx
[root@localhost sbin]#vi /etc/init.d/nginx #参考第 3 章编写的 service 脚本
[root@localhost sbin]#chmod +x /etc/init.d/nginx
[root@localhost sbin]#chkconfig --add nginx
```

2) 配置并启动 Nginx

按照如下操作步骤修改 Nginx 配置文件，启动 Nginx 服务，然后开启防火墙 80 端口。

```
#创建用户"www"和站点目录"/data/www"
[root@localhost sbin]#useradd -s /sbin/nologin -M www
[root@localhost sbin]#mkdir -p /data/www
[root@localhost sbin]#cp ../html/* /data/www
[root@localhost sbin]#chown -R www:www /data/www
[root@localhost sbin]#vi ../conf/nginx.conf
    #配置用户
    user www www;
    #将 server 块修改为如下内容:
    server {
        listen 80;
```

```
        server_name localhost;
        root /data/www;
        index index.html index.htm;
    }
[root@localhost sbin]#service nginx start
[root@localhost sbin]#iptables -I INPUT -p tcp --dport 80 -j ACCEPT
[root@localhost sbin]#service iptables save
```

完成上述操作后，使用浏览器访问 http://192.168.78.11 测试 Nginx 是否启动成功。测试成功后，执行 poweroff 命令关闭服务器，然后准备克隆虚拟机。

3）克隆虚拟机

利用 VMware 基于 1 号服务器克隆出两台虚拟机，作为 2、3 号服务器使用。然后启动 2、3 号服务器，执行前面编写的 netconfig.sh 脚本进行网络配置，将 IP 地址分别设置为 192.168.78.12 和 192.168.78.13。

4）编辑 hosts 文件

在本实验环境中，物理机将作为客户端访问服务器，因此需要在物理机中配置 hosts 文件解析域名。编辑 C:\Windows\System32\drivers\etc\hosts 文件，具体配置如下。

```
192.168.78.11    itshop.test
192.168.78.11    www.itshop.test
192.168.78.12    file.itshop.test
192.168.78.13    upload.itshop.test
```

完成配置后，在浏览器中访问这几个域名，测试配置是否成功。

3. Nginx＋PHP 服务器搭建

完成 Nginx 的安装后，开始搭建 Nginx＋PHP 环境。编号为 3、4、5 的服务器需要安装 PHP，其中 3 号服务器用于提供文件上传服务，包括在客户端上传图片时，对图片进行压缩、生成缩略图、添加水印等处理，最后保存到 6 号的文件存储服务器中；4、5 号服务器是一个集群，用于执行网站的脚本程序。

1）安装 PHP

在 3 号服务器中安装 PHP，具体操作步骤如下。

```
#① 安装依赖包
[root@localhost ~]#yum -y install gcc-c++ libxml2-devel curl-devel \
libjpeg-devel libpng-devel freetype-devel
[root@localhost ~]#tar -zxvf libmcrypt-2.5.8.tar.gz
[root@localhost ~]#cd libmcrypt-2.5.8
[root@localhost libmcrypt-2.5.8]#./configure && make && make install && cd ..
#② 编译安装 PHP
[root@localhost ~]#tar -zxvf php-5.6.27.tar.gz
[root@localhost ~]#cd php-5.6.27
[root@localhost php-5.6.27]#./configure --prefix=/usr/local/php --enable-fpm \
--with-zlib --enable-zip --enable-mbstring --with-mcrypt --with-mysql \
--with-mysqli --with-pdo-mysql --with-gd --with-jpeg-dir --with-png-dir \
```

```
--with-freetype-dir --with-curl --with-openssl --with-mhash --enable-bcmath \
--enable-opcache && make && make install
```

2）部署 Nginx＋PHP 环境

按照如下步骤配置 PHP，启动 PHP-FPM 服务，在 Nginx 配置文件中加入 PHP 支持。

```
#① 复制 php.ini 配置文件
[root@localhost php-5.6.27]#cp php.ini-production /usr/local/php/lib/php.ini
[root@localhost php-5.6.27]#vi /usr/local/php/lib/php.ini #配置时区为 PRC
#② 创建服务脚本、设置开机启动
[root@localhost php-5.6.27]#cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
[root@localhost php-5.6.27]#chmod +x /etc/init.d/php-fpm
[root@localhost php-5.6.27]#chkconfig --add php-fpm
#③ 复制 php-fpm 配置文件、启动服务
[root@localhost php-5.6.27]#cd /usr/local/php/etc
[root@localhost etc]#cp php-fpm.conf.default php-fpm.conf
[root@localhost etc]#vi php-fpm.conf
    #更改"[www]"下的配置
    user=www                              #子进程工作用户
    group=www                             #子进程工作组
    listen=/tmp/php-cgi.sock              #监听 socket 文件
    listen.owner=www                      #socket 文件的所有者
    listen.group=www                      #socket 文件的所属组
[root@localhost etc]#service php-fpm start
#④ 在 Nginx 配置文件中加入 PHP 支持
[root@localhost etc]#vi /usr/local/nginx/conf/nginx.conf
    #在 server 块中进行如下配置：
    index index.html index.htm index.php;
    location ~\.php${
        try_files $uri =404;
        fastcgi_pass unix:/tmp/php-cgi.sock;
        include fastcgi.conf;
    }
[root@localhost etc]#service nginx reload
```

完成上述操作后，Nginx＋PHP 的环境就搭建完成了，可以用 phpinfo 来检查环境是否正常。测试成功后，执行 poweroff 命令关闭服务器，然后准备克隆虚拟机。

3）克隆虚拟机

基于 3 号虚拟机克隆出 4 号和 5 号虚拟机，然后启动虚拟机，利用 netconfig. sh 脚本配置网络，将 IP 地址分别设置为 192. 168. 78. 14 和 192. 168. 78. 15。

4）配置防火墙

由于 4、5 号服务器不需要直接被外部访问，因此可以更改防火墙规则，实现只允许 1 号负载均衡服务器的 IP 地址访问，具体操作如下。

```
#① 查看原有 80 端口规则的序号(即输出结果中第 1 列的数字)
[root@localhost ~]#service iptables status | grep 80
1    ACCEPT     tcp  --  0.0.0.0/0     0.0.0.0/0     tcp dpt:80
#② 修改序号为 1 的规则，利用选项"-s"指定来源 IP 地址为 192.168.78.11
```

```
[root@localhost ~]#iptables -R INPUT 1 -s192.168.78.11 -ptcp --dport 80 -jACCEPT
#③ 保存防火墙配置
[root@localhost ~]#service iptables save
```

完成上述配置后，在物理机中使用浏览器访问 4、5 号服务器进行测试。若无法访问成功，说明防火墙配置生效。

4. 配置反向代理和负载均衡

经过前面的操作，编号为 4、5 的服务器只能由 1 号服务器进行访问。接下来在 1 号服务器中实现反向代理和负载均衡。编辑 Nginx 安装目录下的 conf/nginx.conf 文件，具体配置如下。

```
upstream web_server {
    server 192.168.78.14;
    server 192.168.78.15;
    keepalive 32;                                    #与后端服务器保持的长连接数
}
server {
    listen 80;
    server_name itshop.test www.itshop.test;
    location / {
        proxy_pass http://web_server;
        proxy_http_version 1.1;                      #后端服务器使用 HTTP 1.1
        proxy_set_header Connection "";              #清空客户端 Connection 消息头
        proxy_set_header Host $host;                 #传递请求中的 Host 消息头
        proxy_set_header X-Real-IP $remote_addr;     #传递真实客户端 IP
    }
}
```

上述配置利用 X-Real-IP 请求头传递了真实客户端 IP，为了使后端 4、5 号服务器将来自 1 号服务器的 X-Real-IP 请求头识别为客户端 IP，还需要在 4、5 号服务器的 server 块中添加以下配置。

```
real_ip_header      X-Real-IP;          #从指定消息头获取真实客户端 IP
set_real_ip_from   192.168.78.11;       #只从来自指定 IP 的请求中获取 X-Real-IP
```

完成上述操作并重新加载配置文件后，可以在 4、5 号服务器中创建 phpinfo 文件，然后使用浏览器访问 1 号服务器，反复刷新页面，查看 phpinfo 显示的服务器 IP 和客户端 IP 是否正确。

多学一招：CDN 技术

CDN(Content Delivery Network)是一种新型网络构建方式，主要用于网站加速。它依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率，其运行原理如图 8-8 所示。

CDN 是反向代理的延伸，在结构上实现了多点的冗余，即使某一个节点由于意外发生

故障，对网站的访问能够被自动导向其他健康节点进行响应。但由于 CDN 部署成本高、维护困难，通常大多数中小网站会选择直接从 CDN 提供商购买服务使用。目前，许多知名 CDN 服务提供商提供了静态资源缓存、防御 DDoS(分布式拒绝服务)攻击和 WAF(Web 应用防护系统)等功能，而且使用非常简单，只需将域名的 NS(NameServer)记录设置为 CDN 提供商提供的 DNS 服务器即可。

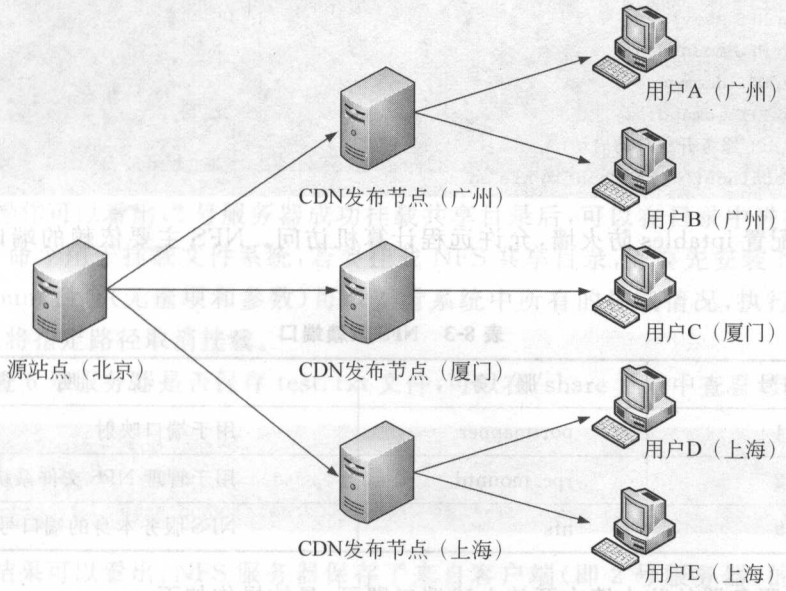


图 8-8　CDN 原理图

8.2.3　搭建 NFS 文件服务器

NFS(Network File System，网络文件系统)是一种用于在网络中共享文件的系统。通过使用 NFS，用户可以用访问本地文件的操作方式来访问远程服务器中的文件。

在前面部署的服务器中，3 号服务器用于将客户端上传的文件保存到 6 号服务器中存储，而 2 号服务器需要将客户端访问的文件从 6 号服务器中取出来并进行缓存。为了使 2、3 号服务器能够同时访问 6 号服务器中的文件，就需要利用 NFS 实现文件共享。

1. 安装和启动 NFS 服务

在 6 号服务器中安装 nfs-utils 软件包即可搭建 NFS 服务，具体命令下。

```
[root@localhost ~]#yum -y install nfs-utils
```

上述命令执行完成后，按照如下操作修改 NFS 的配置文件，配置固定端口号。

```
[root@localhost ~]#vi /etc/sysconfig/nfs
#找到如下一行内容，取消注释
MOUNTD_PORT=892
```

在完成端口号的配置后，即可启动 NFS 服务。由于 NFS 服务依赖于 rpcbind 服务，需

要在开启 NFS 服务前先开启 rpcbind 服务，具体操作如下。

```
#① 开启 rpcbind 服务
[root@localhost ~]#service rpcbind start
Starting rpcbind:                                          [  OK  ]
#② 开启 nfs 服务
[root@localhost ~]#service nfs start
Starting NFS services:                                     [  OK  ]
Starting NFS mountd:                                       [  OK  ]
Starting NFS daemon:                                       [  OK  ]
Starting RPC idmapd:                                       [  OK  ]
#③ 配置 NFS 服务开机自动启动
[root@localhost ~]#chkconfig nfs on
```

接下来配置 iptables 防火墙，允许远程计算机访问。NFS 主要依赖的端口号如表 8-3 所示。

表 8-3 NFS 依赖端口

端口号	服 务	说 明
111	portmapper	用于端口映射
892	rpc. mountd	用于管理 NFS 文件系统
2049	nfs	NFS 服务本身的端口号

在 NFS 服务器的防火墙中开放上述端口即可，具体操作如下。

```
[root@localhost ~]#iptables -I INPUT -p udp --dport 111 -j ACCEPT
[root@localhost ~]#iptables -I INPUT -p udp --dport 892 -j ACCEPT
[root@localhost ~]#iptables -I INPUT -p tcp --dport 2049 -j ACCEPT
[root@localhost ~]#service iptables save
```

2. 配置共享目录

利用 NFS 可以将某个指定目录配置为共享目录，允许外部计算机访问，具体操作如下。

```
#① 创建用于共享的目录
[root@localhost ~]#mkdir /share
#② 将目录权限设为 777，从而允许远程客户端操作
[root@localhost ~]#chmod 777 /share
#③ 配置/share 为共享目录，语法是"路径  IP 段(权限)"，任意 IP 用"*"表示
[root@localhost ~]#echo '/share *(rw)' >/etc/exports
#④ 重新加载 NFS 共享配置，使更改后的配置生效
[root@localhost ~]#service nfs reload
```

完成上述配置后，在 2 号服务器中挂载 NFS 共享目录，实现远程文件的读写操作，具体如下。

```
#① 安装 NFS 软件包
[root@localhost ~]#yum -y install nfs-utils
```

```
#② 查看 NFS 服务器中的共享目录
[root@localhost var]# showmount -e 192.168.78.16
Export list for 192.168.78.16:
/share *
#③ 将 NFS 服务器共享的"/share"目录挂载到本地目录"/data/share"(也可以是其他目录)
[root@localhost ~]#mkdir /data/share
[root@localhost ~]#mount 192.168.78.16:/share /data/share
#④ 读写文件测试
[root@localhost ~]#cd /data/share
[root@localhost share]#echo Hello >test.txt
[root@localhost share]#cat test.txt
Hello
```

从上述操作可以看出，2 号服务器成功挂载共享目录后，可以在目录中进行文件读写。其中，mount 命令用于挂载文件系统，若要挂载 NFS 共享目录，需要先安装 NFS 软件包。直接执行 mount 命令(无选项和参数)可以查看系统中所有的挂载情况，执行“umount 路径”命令可以将指定路径取消挂载。

为了检查 6 号服务器是否保存 test. txt 文件，可以在/share 目录中查看，具体如下。

```
[root@localhost ~]#cd /share
[root@localhost share]#ll test.txt
-rw-r--r--.  1  nfsnobody  nfsnobody  6 Nov 15 15:11  test.txt
```

从上述结果可以看出，NFS 服务器保存了来自客户端(即 2 号服务器)的 text. txt 文件，且该文件所属的用户和用户组都是 nfsnobody。nfsnobody 是在安装 NFS 软件包后自动创建的用户，在/etc/passwd 文件中可以查看该用户的详细信息，如下所示。

```
[root@localhost ~]#cat /etc/passwd | grep nfsnobody
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
```

另外，由于 mount 命令只能临时挂载目录，重启后失效。为了使 2 号服务器开机后自动挂载共享目录，需要在系统的/etc/fstab 文件中，配置开机时自动读取的文件系统信息。根据该文件的语法，通过如下命令新增一行配置，实现自动挂载 NFS 目录。

```
[root@localhost ~]#echo '192.168.78.16:/share /data/share nfs defaults 0 0' \
>>/etc/fstab
```

上述操作执行后，可以实现当服务器重新启动后，自动挂载 NFS 远程文件。

3. 配置文件缓存服务器

许多网站都提供文件的上传和下载功能。例如，一个电子商务网站需要存储大量的商品图片，这些图片是由网站后台的工作人员在编辑商品信息时上传的，每当客户浏览商品信息时就会将这些图片展示出来。当网站的规模和访问量越来越大时，对于经常访问的文件进行缓存就显得非常重要。下面将在 2 号服务器中配置 Nginx 缓存功能，通过部署文件缓存服务器降低后端文件存储服务器的压力。

打开Nginx的配置文件，在http块中添加如下配置，定义缓存的基本规则和保存目录。

```
proxy_temp_path proxy_temp;
proxy_cache_path proxy_cache levels=1:2 keys_zone=one:80m inactive=7d max_size=5g;
```

上述配置将缓存区内存空间设置为80MB，磁盘空间为5GB，自动清理7天内未被访问的缓存文件。此处可以根据服务器实际的硬件能力和具体需求进行配置。

接下来在server块中添加具体的缓存配置，具体如下。

```
server {
    listen 80;
    server_name file.itshop.test;
    add_header X-Cache $upstream_cache_status;
    location / {
        proxy_pass http://web_server;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_cache one;
        proxy_cache_key $host$uri$is_args$args;
        proxy_cache_valid 200 304 2d;
    }
}
upstream web_server {
    server localhost:81;
    keepalive 32;
}
server {
    listen 81;
    server_name localhost;
    root /data/share;
    index index.html index.htm;
}
```

完成上述配置后，Nginx即可从NFS共享目录中读取文件缓存到本地。在使用浏览器访问测试时，通过HTTP响应头X-Cache即可获知请求的文件是否被成功缓存。

另外，当服务器更新了一些CSS、JS文件时，为了避免缓存导致无法即时生效，可以在链接中加上URL参数保存版本号，从而在版本更新时重新缓存。示例HTML代码如下。

```
<link rel="stylesheet" type="text/css" href="style.css?ver=1.0" />
<script type="text/javascript" src="common.js?ver=20161118"></script>
```

4. 配置文件上传服务器

在完成文件下载服务器后，接下来将3号服务器配置为文件上传服务器，实现接收用户上传的文件保存到共享目录中。下面通过具体操作步骤进行讲解。

1）挂载NFS共享目录

按照之前2号服务器的挂载方式，为3号服务器挂载共享目录，具体步骤如下。

```
#① 安装 NFS 软件包
[root@localhost ~]#yum -y install nfs-utils
#② 创建目录并进行挂载
[root@localhost ~]#mkdir /data/share
[root@localhost ~]#mount 192.168.78.16:/share /data/share
#③ 实现开机自动挂载
[root@localhost ~]#echo '192.168.78.16:/share /data/share nfs defaults 0 0' \
>>/etc/fstab
```

2）配置 Nginx 对请求数据量的限制

Nginx 支持对客户端请求时发送的数据量进行限制，在默认情况下只允许 1MB 的数据，如果超过了这个大小将会返回 413(请求实体过大)错误。通过 client_max_body_size 指令可以更改这个限制，该指令可以用于 http、server、location 块中。

接下来，在 Nginx 配置文件的 server 块中将最大数据量提高到 20MB，具体配置如下。

```
client_max_body_size 20m;
```

另外，还需要注意 php.ini 中对于上传文件的限制，将限制增加到 10M。

```
[root@localhost ~]#vi /usr/local/php/lib/php.ini
post_max_size=20M                    #通过 POST 提交的最大限制
file_uploads=On                      #是否允许文件上传
upload_max_filesize=10M              #上传文件最大限制
;upload_tmp_dir =                   #上传文件临时保存目录(默认使用/tmp 目录)
```

完成上述配置的更改后，执行如下命令使配置生效。

```
[root@localhost ~]#service php-fpm reload
[root@localhost ~]#service nginx reload
```

3）上传文件测试

在站点目录下编写一个 upload.php 文件进行上传测试，文件内容如下。

```
1   <form method="post" enctype="multipart/form-data">
2       <input type="file" name="up">
3       <input type="submit">
4   </form>
5   <?php
6   if(isset($_FILES['up'])&& $_FILES['up']['error'] ==0){
7       $savepath='uploads/'.time().'.dat';
8       if(move_uploaded_file($_FILES['up']['tmp_name'], "/data/share/$savepath")){
9           echo "File: <a href=\"http://file.itshop.test/$savepath\">Download</a>";
10      }
11  }
```

上述代码中，第 1～4 行用于显示文件上传表单，第 5～11 行用于判断当前请求是否有上传文件，如果有则将文件保存到/data/share/uploads 目录中，并以时间戳作为文件名。

为了确保 PHP 保存上传文件，需要预先创建 uploads 目录并设置 777 权限。

```
[root@localhost ~]#cd /data/share
[root@localhost share]#mkdir -m 777 uploads
```

接下来在浏览器中访问文件上传页面，选择文件进行上传，如图 8-9 所示。

图 8-9 测试文件上传

从图 8-9 中可以看出，上传文件后 PHP 显示了文件的下载地址，单击链接即可下载文件。

8.2.4 搭建 MySQL 数据库服务器

MySQL 数据库是一个开放源代码的关系型数据库管理系统，具有低成本、跨平台、高性能等特点，在 Web 应用领域广受欢迎。本节将基于 MySQL 5.5 社区版（MySQL Community Server 5.5）进行安装和使用，在官方网站 https://www.mysql.com 可以获取软件的下载，如图 8-10 所示。

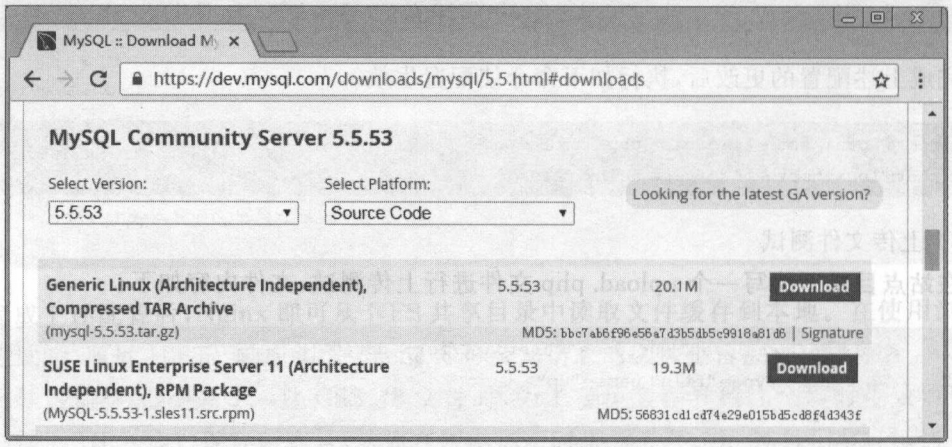


图 8-10 获取 MySQL 数据库

1. 安装和启动 MySQL 服务

1）编译安装 MySQL

将 MySQL 源码包下载到之前部署的 7 号服务器中，然后通过如下步骤进行安装。

```
#① 安装依赖包
[root@localhost ~]#yum -y install gcc-c++ cmake ncurses-devel
#② 解压文件，编译安装 MySQL
```

```
[root@localhost ~]#tar -zxvf mysql-5.5.53.tar.gz
[root@localhost ~]#cd mysql-5.5.53
[root@localhost mysql-5.5.53]#cmake -DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_general_ci
[root@localhost mysql-5.5.53]#make && make install && cd ..
```

2）配置 MySQL

MySQL 默认配置文件位于/etc/my. cnf，在 MySQL 安装后需要配置数据保存目录、sock 文件保存目录和工作用户。具体操作如下。

```
#① 编辑 MySQL 的配置文件
[root@localhost ~]#vi /etc/my.cnf
    #找到如下配置进行更改
    datadir=/data/mysql                    #数据保存目录
    socket=/tmp/mysql.sock                 #sock 文件保存目录
    user=mysql                             #MySQL 的工作用户
#② 根据 my.cnf 中的配置，创建 mysql 用户
[root@localhost ~]#useradd -s /sbin/nologin -M mysql
```

在完成配置后，执行 MySQL 中的 mysql_install_db 程序初始化数据库，初始化后将会在数据保存目录中生成数据库文件。具体操作如下。

```
[root@localhost ~]#cd /usr/local/mysql
[root@localhost mysql]#./scripts/mysql_install_db
```

3）启动 MySQL 并添加到服务

完成 MySQL 的配置和数据库初始化后，即可启动 MySQL 服务。具体操作如下。

```
[root@localhost mysql]#cp support-files/mysql.server /etc/init.d/mysql
[root@localhost mysql]#chkconfig --add mysql
[root@localhost mysql]#service mysql start
```

MySQL 默认监听 3306 端口。启动 MySQL 后，可以用如下命令查看 MySQL 正在监听的端口。

```
[root@localhost ~]#netstat -tnlp | grep mysql
tcp    0    0  0.0.0.0:3306    0.0.0.0:*       LISTEN      15919/mysqld
```

另外，为了允许集群中的其他服务器通过 3306 端口访问 MySQL 服务器，还需要配置防火墙规则，开放 3306 端口，具体操作如下。

```
[root@localhost ~]#iptables -I INPUT -p tcp --dport 3306 -j ACCEPT
[root@localhost ~]#service iptables save
```

2. 配置 MySQL 中的用户

MySQL 数据库管理系统通过用户身份机制来管理数据库。在安装数据库后，MySQL 提供了一个拥有最高权限的 root 用户，在默认情况下并没有登录密码，且允许匿名登录。

为了保护数据库的安全，接下来对 MySQL 中的用户进行配置。

```
#① 启动 MySQL 客户端工具，登录数据库
[root@localhost ~]#cd /usr/local/mysql/bin
[root@localhost bin]#./mysql
#② 在 MySQL 客户端中依次输入如下 SQL 语句执行：
mysql>UPDATE mysql.user SET password=password('123456')WHERE user='root';
mysql>DELETE FROM mysql.user WHERE user='';
mysql>FLUSH PRIVILEGES;
mysql>EXIT
#③ 完成上述操作后，输入用户名 root 和密码 123456 重新登录
[root@localhost bin]#./mysql -uroot -p123456
mysql>EXIT
```

在上述操作中，第 2 步登录 MySQL 客户端后依次执行的 4 条 SQL 语句，分别用于设置 root 用户的密码、删除匿名用于、更新权限和退出客户端。

3. 实现 MySQL 主从复制

对于 LNMP 环境，最简单的部署方案是将 Linux、Nginx、MySQL、PHP 全部安装在一台服务器中，但这样的方案只适合小型网站。当网站中的数据达到一定规模以后，数据库最容易产生瓶颈。为了在集群环境中减轻数据库方面的压力，可以部署多台 MySQL 服务器实现主从复制。

1）克隆虚拟机

在 7 号服务器中完成 MySQL 的安装后，克隆出 8 号服务器。克隆后启动 8 号服务器，执行 netconfig.sh 脚本配置网络，将 IP 地址设置为 192.168.78.18。

2）主服务器开启 bin 日志

MySQL 的 bin 日志（二进制日志）功能用于记录数据发生的改变，可以用于数据库的增量备份、数据库之间的复制等操作。在 7 号服务器中执行 vi /etc/my.cnf 命令编辑 MySQL 的配置文件，在[mysqld]节中添加如下配置，开启主服务器的 bin 日志功能。

```
log-bin=mysqlbin-log
server-id=17
```

上述配置中，第 1 行表示创建 bin 日志并将日志文件命名为 mysqlbin-log，第 2 行表示服务器的唯一 ID，此处用 IP 地址的最后一位作为 ID（也可以用其他值）。

在更改配置文件后，执行 service mysql restart 重启 MySQL 服务，然后在 MySQL 的数据目录下即可查看 bin 日志文件，如下所示。

```
[root@localhost ~]#cd /data/mysql
[root@localhost mysql]#ll | grep mysqlbin
-rw-rw----. 1 mysql mysql      107 Nov 16 15:39 mysqlbin-log.000001
-rw-rw----. 1 mysql mysql       22 Nov 16 15:39 mysqlbin-log.index
```

上述文件是数据库生成的操作日志，这些文件由 MySQL 自动管理。当需要和其他服务器数据同步时，会通过这些日志来读取数据库的更改记录。

3）创建用于主从复制的用户

实现主从复制，需要“从服务器”(8 号)获取“主服务器”(7 号)中的数据以实现数据同步，“从服务器”在获取数据时必须要使用一个用户账号来登录“主服务器”。接下来在“主服务器”中创建一个专门用于“从服务器”登录的用户账户，具体如下。

```
#① 登录 MySQL 服务器
[root@localhost ~]#/usr/local/mysql/bin/mysql -uroot -p123456
#② 执行如下 SQL 语句创建用户，分配权限
#用户名 slave，密码 123456，仅客户端 192.168.78.18 可登录，"*.*"表示可操作所有的数据库和表
mysql>GRANT REPLICATION SLAVE ON *.* TO 'slave'@'192.168.78.18'
IDENTIFIED BY '123456';
```

4）查看主服务器 bin 日志状态

在 MySQL 客户端程序中执行 SHOW MASTER STATUS 语句查看当前 bin 日志的状态，用于指定“从服务器”为了同步数据而读取“主服务器”二进制日志的位置。执行结果如下。

```
mysql>SHOW MASTER STATUS;
+----------------------+----------+---------------+------------------+
| File                 | Position | Binlog_Do_DB  | Binlog_Ignore_DB |
+----------------------+----------+---------------+------------------+
| mysqlbin-log.000001  |      262 |               |                  |
+----------------------+----------+---------------+------------------+
```

上述输出结果中，File 字段表示日志文件的名称，Position 字段表示当前记录的位置。配置主从同步时，需要在“从服务器”中指定这些信息来确定同步的日志文件和位置。

5）配置从服务器

在 8 号服务器中编辑/etc/my.cnf 配置文件，配置“从服务器”的唯一 ID。

```
server-id=18
```

在完成配置后，执行 service mysql restart 命令使配置生效。然后使用 MySQL 客户端工具登录“从服务器”，执行如下 SQL 语句实现“从服务器”自动同步“主服务器”。具体操作如下。

```
mysql>CHANGE MASTER TO master_host='192.168.78.17', master_user='slave',
master_password='123456', master_log_file='mysqlbin-log.000001', master_log_pos=262;
mysql>START SLAVE;
mysql>SHOW SLAVE STATUS \G
*************************** 1. row ***************************
      Slave_IO_State: Waiting for master to send event    #等待主服务器发送事件
         Master_Host: 192.168.78.17                       #主服务器地址
         Master_User: slave                               #主服务器用户名
         Master_Port: 3306                                #主服务器端口号
       Connect_Retry: 60                                  #连接失败重试次数
     Master_Log_File: mysqlbin-log.000001                 #主服务器日志文件
 Read_Master_Log_Pos: 262                                 #主服务器日志读取位置
      Relay_Log_File: localhost-relay-bin.000002
```

```
          Relay_Log_Pos: 256
    Relay_Master_Log_File: mysqlbin-log.000001
        Slave_IO_Running: Yes                          #从服务器 IO 运行中
       Slave_SQL_Running: Yes                          #从服务器 SQL 运行中
⋮
```

在上述输出结果中，若看到 Slave_IO_Running 和 Slave_SQL_Running 两项的结果为 yes，则说明当前已经配置成功。另外，若要停止“从服务器”的同步，可以执行 SLAVE STOP 语句。

6）测试同步情况

经过以上操作，已经完成了“从服务器”自动同步“主服务器”。由于这两台服务器用于实现读写分离，由“主服务器”负责写数据的操作，“从服务器”负责读数据的操作，因此不需要“主服务器”同步“从服务器”中的数据。接下来对两台服务器分别进行读写，测试同步情况。

```
#① 在"主服务器"中写入数据
mysql>CREATE TABLE test.a(id INT, num INT);
mysql>INSERT INTO test.a VALUES(1, 2);
#② 在"从服务器"中读取数据
mysql>SELECT * FROM test.a \G
id: 1
num: 2
#③ 在"从服务器"中写入数据
mysql>INSERT INTO test.a VALUES(3, 4);
#④ 在"主服务器"中读取数据
mysql>SELECT * FROM test.a \G
id: 1
num: 2
```

上述操作在“主服务器”的 test 数据库中创建了数据表 a，并插入一条测试数据(id=1，num=2)；为了测试“从服务器”是否已经同步，在“从服务器”中查询 test 数据库中的 a 表，可以看到“主服务器”插入的测试数据。反之，若在“从服务器”中插入测试数据，则在“主服务器”中无法查询到。

以上操作仅用于测试，在实际使用时，应该保证两个数据库中数据是一致性的。

8.2.5 搭建 Memcached 缓存服务器

Memcached 是一个高性能的分布式内存对象缓存系统。在 Web 应用中，使用 Memcached 可以利用内存来缓存一些被频繁访问的数据，从而减少 MySQL 数据库的查询次数。相对于基于磁盘的缓存，内存缓存具有非常高的读写性能，但无法持久保存数据，因此 Memcached 适合保存一些经常访问但不重要的数据，即使数据丢失也不会对业务造成影响。

在 Memcached 官方网站 http://memcached.org 可以获取软件的下载地址，如图 8-11 所示。

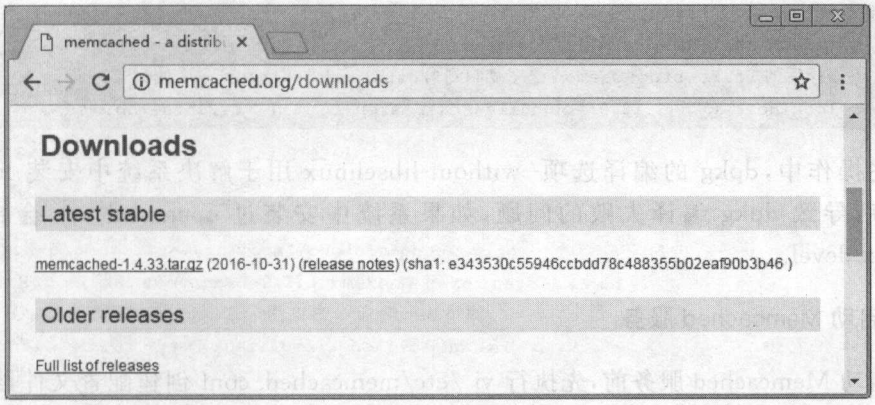


图 8-11　获取 Memcached

1. 编译安装 Memcached

在将 Memcached 下载到 9 号服务器后，按照如下操作步骤进行安装。

```
#① 安装依赖包
[root@localhost ~]#yum -y install gcc libevent-devel
#② 编译安装 Memcached
[root@localhost ~]#  tar -zxvf memcached-1.4.33.tar.gz
[root@localhost ~]#  cd memcached-1.4.33
[root@localhost memcached-1.4.33]#./configure
[root@localhost memcached-1.4.33]#make && make install
#③ 添加到系统服务、配置开机自动启动
[root@localhost memcached-1.4.33]#cd scripts
[root@localhost scripts]#cp memcached-init /etc/init.d/memcached
[root@localhost scripts]#chkconfig --add memcached
#④ 在 memcached 服务脚本中还调用了 start-memcached 脚本，需要复制该脚本文件到指定路径
[root@localhost scripts]#mkdir -p /usr/share/memcached/scripts
[root@localhost scripts]#cp start-memcached /usr/share/memcached/scripts/
[root@localhost scripts]#cd ~
#⑤ 为 start-memcached 脚本中指定的路径创建链接
[root@localhost ~]#ln -s /usr/local/bin/memcached /usr/bin/memcached
```

完成以上操作后，目前还无法通过 service 命令启动 memcached 服务，这是因为 memcached 中的脚本依赖于 perl 和 start-stop-daemon，其中 start-stop-daemon 是 Debain 系列发行版 Linux 系统中提供的命令，CentOS 系统需要单独下载安装才可以使用，具体操作步骤如下。

```
#① 从 debain 镜像中获取 dpgk(Debian Packager)
[root@localhost ~]#yum -y install gcc-c++wget perl xz ncurses-devel
[root@localhost ~]#wget \
http://ftp2.cn.debian.org/debian/pool/main/d/dpkg/dpkg_1.18.9.tar.xz
#② 单独编译 start-stop-daemon
[root@localhost ~]#tar -xvf  dpkg_1.18.9.tar.xz
```

```
[root@localhost ~]#cd dpkg-1.18.9
[root@localhost dpkg-1.18.9]#./configure --without-libselinux && make
#③ 将编译后的 start-stop-daemon 程序复制到"/usr/local/sbin/"目录中使用
[root@localhost dpkg-1.18.9]#cp utils/start-stop-daemon /usr/local/sbin/
```

上述操作中，dpkg 的编译选项--without-libselinux 用于解决系统中安装 libselinux-devel 之后导致 dpkg 编译失败的问题，如果系统中安装过 openssl-devel，会自动安装 libselinux-devel。

2. 启动 Memcached 服务

在启动 Memcached 服务前，先执行 vi /etc/memcached.conf 创建配置文件，编写内容如下。

```
-m 512                  #分配的内存大小,单位是 MB,默认 64MB
-p 11211                #配置监听的 TCP 端口,默认为 11211
-u nobody               #配置 Memcached 的工作用户
-c 1024                 #配置最高并发连接数,默认 1024
-t 16                   #配置使用的线程数,默认 4
```

上述配置中的选项可以通过/usr/local/bin/memcached -h 命令查看详细说明，根据服务器的硬件能力合理配置即可。

保存配置文件后，执行 service memcached start 命令启动 memcached 服务。另外，为了使其他服务器能够访问 Memcached，需要配置防火墙，开放 11211 端口。

```
[root@localhost ~]#iptables -I INPUT -p tcp --dport 11211 -j ACCEPT
[root@localhost ~]#service iptables save
```

3. PHP 访问 Memcached

为了使 PHP 能够访问 Memcached，需要在 PHP 中安装相应的扩展。PHP 的 Memcached 扩展可以在网址为 http://pecl.php.net 的 PECL（The PHP Extension Community Library，PHP 社区扩展库）网站中获取，而该扩展又依赖于 libmemcached，在 http://libmemcached.org 网站中获取。

将 PHP 的 Memcached 扩展和 libmemcached 下载到 3、4、5 号服务器中，按照以下步骤进行安装。

```
#① 安装依赖包
[root@localhost ~]#yum -y install cyrus-sasl-devel
#② 编译安装 libmemcached
[root@localhost ~]#tar -zxvf libmemcached-1.0.18.tar.gz
[root@localhost ~]#cd libmemcached-1.0.18
[root@localhost libmemcached-1.0.18]#./configure && make && make install && cd ..
#③ 为 PHP 的 Memcached 扩展生成 configure 文件
[root@localhost ~]#tar -zxvf memcached-2.2.0.tgz
[root@localhost ~]#cd memcached-2.2.0
```

```
[root@localhost memcached-2.2.0]#/usr/local/php/bin/phpize
Configuring for:
PHP Api Version:         20131106
Zend Module Api No:      20131226
Zend Extension Api No:   220131226
#④ 编译安装 PHP 的 Memcached 扩展
[root@localhost memcached-2.2.0]#./configure \
--with-php-config=/usr/local/php/bin/php-config
[root@localhost memcached-2.2.0]#make && make install && cd ..
#⑤ 在 PHP 的配置文件 php.ini 中加载 Memcached 扩展
[root@localhost ~]#vi /usr/local/php/lib/php.ini
#添加如下配置:
extension=/usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/memcached.so
#⑥ PHP-FPM 重新加载配置
[root@localhost ~]#service php-fpm reload
```

完成上述操作后，PHP 的 Memcached 扩展已经安装完成并启用，通过 phpinfo 可以查看该扩展的详细信息。也可以通过编写如下代码测试 PHP 是否能够访问 Memcached 服务器。

```
<?php
#连接 Memcached 服务器
$mem=new Memcached();
$mem->addServer('192.168.78.19', 11211);
#保存数据(Key/Value 形式,Key=UserName,Value=James)
$mem->set('UserName', 'James');
#获取数据(根据 Key=UserName,获得 Value)输出结果:James
echo $mem->get('UserName');
```

将上述代码以 test.php 文件名保存到站点文档目录下，然后通过浏览器访问进行测试即可。若运行结果是 James，则说明 Memcached 可以使用。

8.2.6　ThinkPHP 项目部署

在完成 LNMP 分布式集群搭建后，为了使 Web 应用能够在集群中工作，还需要进行一些必要的环境配置。本节通过部署一个基于 ThinkPHP 框架开发的电子商务网站为例，讲解如何将项目部署到集群中。读者可以通过本书的配套源代码资源获取该项目的所有文件。

1. 创建 MySQL 独立数据库用户

一个 MySQL 服务器中可以管理多个数据库，并且 MySQL 将自身的一些配置、用户账号、运行信息等也保存在数据库中。大多数 Web 项目在开发时都会遵循一个项目只访问一个数据库的习惯，因此可以在 MySQL 中创建一个数据库，并提供一个只能访问该数据库的用户。具体操作如下。

```
#① 在 7 号服务器中启动 MySQL 客户端,登录 MySQL 服务器
```

```
[root@localhost ~]#/usr/local/mysql/bin/mysql -uroot -p123456
#② 创建数据库 itshop
mysql>CREATE DATABASE itshop;
#③ 为 4、5 号服务器创建无权限用户 itshop,密码为 123456
mysql>GRANT USAGE ON *.* TO 'itshop'@'192.168.78.14' IDENTIFIED BY '123456';
mysql>GRANT USAGE ON *.* TO 'itshop'@'192.168.78.15' IDENTIFIED BY '123456';
#④ 为用户分配指定数据库的权限
mysql>GRANT ALL PRIVILEGES ON itshop.* TO 'itshop'@'192.168.78.14';
mysql>GRANT ALL PRIVILEGES ON itshop.* TO 'itshop'@'192.168.78.15';
#⑤ 更新权限、退出客户端
FLUSH PRIVILEGES;
EXIT
```

完成上述操作后,即可在 4、5 号两台 Nginx+PHP 服务器中访问 MySQL 数据库。从安全角度考虑,上述操作在分配权限时遵循了最小权限原则,即为用户只赋予其完成的操作所必需的权限,这样可以在发生意外时将风险降到最低。

2. 上传项目文件到站点目录

本书的配套源代码中提供了一个基于 ThinkPHP 3.2.3 框架开发的电子商务网站项目。将项目的压缩包上传到 3、4、5 号服务器中,解压到站点目录下,然后查看目录结构,具体操作如下。

```
#① 解压项目压缩包
[root@localhost ~]#tar -zxvf itshop-1.0.tar.gz
#② 清理站点目录,部署项目文件,设置权限
[root@localhost ~]#rm -rf /data/www
[root@localhost ~]#mv itshop-1.0 /data/www
[root@localhost ~]#chown -R www:www /data/www
#③ 查看项目目录结构
[root@localhost ~]#ls /data/www
 Application                    #项目的业务代码目录
 composer.json                  #用于 composer(PHP 项目依赖管理器)的文件
 data.sql                       #用于导入项目数据库的文件(导入后应删除此文件)
 index.php                      #项目入口文件
 upload.php                     #提供文件上传接口的文件
 Public                         #可公开访问的目录
 README.md                      #说明文件(Markdown 格式)
 ThinkPHP                       #ThinkPHP 框架目录
```

3. 配置项目数据库

在项目的目录中,data.sql 是用于导入数据库的 SQL 文件,该文件保存了项目所需要创建的各种数据表。接下来将 data.sql 文件下载到 7 号服务器,然后向数据库中导入该文件,具体操作如下。

```
#① 将 data.sql 下载到 7 号服务器
[root@localhost ~]#curl -o data.sql http://192.168.78.13/data.sql
```

```
[root@localhost ~]#ll data.sql
-rw-r--r--. 1 root root 2939 Nov 17 05:49 data.sql
#② 登录 MySQL 客户端
[root@localhost ~]#/usr/local/mysql/bin/mysql -uroot -p123456
#③ 创建选择之前创建的数据库
mysql>USE itshop;
#④ 导入 data.sql 文件
mysql>source /root/data.sql
```

完成项目数据库导入后，在 4、5 号服务器中对项目进行配置，从而能够访问数据。ThinkPHP 框架支持分布式数据库和读写分离，只需参考官方手册简单配置即可。具体操作步骤如下。

```
#① 删除 data.sql 文件,防止对外泄露
[root@localhost ~]#cd /data/www
[root@localhost www]#rm -f data.sql
#② 更改项目配置文件,配置分布式数据库
[root@localhost www]#vi Application/Common/Conf/config.php
  #找到如下配置进行修改(若没有则添加)
  'DB_DEPLOY_TYPE' =>1,                          #开启分布式数据库支持
  'DB_RW_SEPARATE' =>TRUE,                       #读写分离
  'DB_TYPE' =>'MYSQL',                           #数据库类型
  'DB_HOST' =>'192.168.78.17,192.168.78.18',     #服务器地址
  'DB_NAME' =>'itshop',                          #数据库名
  'DB_USER' =>'itshop',                          #用户名
  'DB_PWD' =>'123456',                           #密码
  'DB_PORT' =>'3306',                            #端口
  'DB_PREFIX' =>'shop_',                         #数据库表前缀
  'DB_CHARSET' =>'utf8',                         #数据库编码
```

完成上述操作后，项目就可以运行了。通过浏览器访问 http://www.itshop.test 进行测试，网站首页的访问结果如图 8-12 所示。

4. 单一入口配置

ThinkPHP 是一种典型的单入口程序，所谓单入口是指站点文档目录下只有一个 index.php 提供动态请求访问，具体的功能名称通过 URL 参数或 PATHINFO 进行传递。下面演示本项目的 URL 地址。

```
#以下四种方式都可以访问 admin 模块 login 控制器 index 操作
#① URL 参数方式,指定文件名
http://www.itshop.test/index.php?m=admin&c=login&a=index
#② URL 参数方式,省略文件名
http://www.itshop.test/?m=admin&c=login&a=index
#③ PATHINFO 方式,指定文件名
http://www.itshop.test/index.php/Admin/Login/index
#④ PATHINFO 方式,伪静态
http://www.itshop.test/Admin/Login/index.html
```

图 8-12 项目运行效果

上述示例中，模块、控制器和操作都是 ThinkPHP 框架设计的机制，用于告知 index.php 入口程序当前用户访问的具体页面或功能。其中，第 1 种方式对环境没有特殊要求；第 2 种方式需要配置 Nginx 的 index 指令使 index.php 作为默认页面，并注意其与 index.html、index.htm 的优先级；第 3、4 种方式需要对 Nginx 进行更深入的配置，将文件不存在的地址内部重定向到 index.php。

为了实现第 3、4 种 URL 地址的需求，可以利用 Nginx 的 try_files 和 rewrite 指令。接下来在 4、5 号服务器中编辑 Nginx 配置文件，在 server 块中重新进行 PHP 相关的配置。

```
#针对 PATHINFO 方式配置重写规则
rewrite ^/index.php/(.*)/index.php?s=$1 break;
#将文件不存在的请求内部重定向到 index.php
location / {
    try_files $uri $uri/ /index.php?s=$uri;
}
#防止静态文件不存在的请求被内部重定向到 index.php
location ~\.(gif|jpg|jpeg|png|bmg|swf|xml|ico|css|js|map|txt)${
    expires 30d;
}
#将".php"文件交给 PHP-FPM 处理
location ~\.php${
    try_files $uri =404;
    fastcgi_pass unix:/tmp/php-cgi.sock;
    include fastcgi.conf;
}
```

完成上述配置后，通过浏览器访问网站后台的 URL 地址 http://www.itshop.test/Admin 进行测试，由于此时还没有登录后台管理系统，网站会自动跳转到后台登录页面，如

图 8-13 所示。

图 8-13　后台访问测试

5. 利用 Memcached 保存 Session

Session 是一种会话技术，主要用于实现用户登录功能，使服务器能够区分每个请求所对应的用户。PHP 中的 Session 机制基于 Cookie，Cookie 是浏览器和服务器之间来回发送的一段数据，服务器可以通过 Set-Cookie 响应头将数据发送给浏览器保存，而浏览器以后每次请求都会将该数据放入 Cookie 请求消息头中发送。

当服务器端 PHP 开启 Session 时，每收到一个新客户端浏览器的请求，就会为这个浏览器创建一个 Session 文件保存在服务器中，其文件名是一串自动生成的密钥（Session Key），服务器利用 Set-Cookie 将密钥响应给浏览器，浏览器下次请求就会携带 Cookie 中的密钥进行发送。由于服务器为不同浏览器生成的密钥不同，且难以伪造，因此就可以区分当前请求是哪一个浏览器发出的。网站用户登录功能的原理就是将用户 ID 保存到 Session 文件中，从而区分每个请求对应的用户。如果通过浏览器提交的密钥找不到 Session 文件，就说明该用户没有登录。

在集群环境中，Nginx＋PHP 服务器有多个，功能都是相同的，但不同服务器保存的 Session 文件不同，这就导致服务器无法正确识别每个请求对应的用户。为了解决这个问题，需要两台服务器能够共享 Session 数据，而利用 Memcached 保存 Session 就是一种非常高效的方案。

下面在 3、4、5 号服务器中利用 .user. ini 配置文件针对项目开启 Session 转存

Memcached。

```
#创建 PHP 分布式配置文件(可以用来修改 php.ini 中的一些非关键配置)
[root@localhost ~]#vi /data/www/.user.ini
#编写内容具体如下:
session.save_handler=memcached
session.save_path="192.168.78.19:11211"
```

完成上述操作后,可以执行 rm -f /tmp/sess_* 命令删除服务器中的 Session 文件,然后通过浏览器访问网站进行测试,多次刷新后,如果在/tmp 目录下没有生成新的文件名以 sess_开始的文件,则说明 Session 已经保存到 Memcached 中。另外,.user.ini 文件中的配置可以自动生效,但可能会有延迟,为了避免影响测试结果,可以执行 service php-fpm reload 命令使配置立即生效。

接下来就可以在网站中注册新用户进行测试,或通过用户名 admin 密码 123456 登录网站的后台进行测试。如果能够顺利登录成功,说明配置正确。

6. 文件上传下载分离

在项目中,Public 目录用于存放网站的一些静态文件,Public 中的 Upload 目录用于存放用户上传的图片文件。在前面的环境搭建中,已经部署了 2、3、6 号服务器专门用于文件下载、上传和存储,接下来就将 Public 目录转移到 6 号服务器,并将网站中的静态文件链接替换为 2 号服务器的地址,然后将文件上传接口部署到 3 号服务器中。具体操作步骤如下。

```
#① 将项目上传到 2 号服务器,解压后复制 Public 到 NFS 目录中
[root@localhost ~]#tar -zxvf itshop-1.0.tar.gz
[root@localhost ~]#cp -R itshop-1.0/Public /data/share
#② 在 4、5 号服务器中配置项目的模板路径
[root@localhost ~]#vi /data/www/Application/Common/Conf/config.php
    #修改配置如下:
    'TMPL_PARSE_STRING' =>array(
        '__PUBLIC__' =>'//file.itshop.test/Public',
        '__UPLOAD_API__' =>'//upload.itshop.test/upload.php',
    ),
#③ 在 3 号服务器中,删除上传目录,然后将 NFS 共享目录链接为上传目录
[root@localhost ~]#cd /data/www
[root@localhost www]#rm -rf data.sql Public/Uploads
[root@localhost www]#ln -s /data/share/Public/Uploads Public/Uploads
[root@localhost www]#chmod -R 777 Public/Uploads
#④ 编辑 3 号服务器的 Nginx 配置文件,限制只允许访问文件上传接口
[root@localhost www]#vi /usr/local/nginx/conf/nginx.conf
    #按照如下方式配置
    location / {
        return 403;
    }
    location /upload.php {                                  #此行替换原来的"location ~\.php${"
        try_files $uri =404;
        add_header Access-Control-Allow-Origin *;           #添加此行允许跨域请求
        fastcgi_pass unix:/tmp/php-cgi.sock;
        include fastcgi.conf;
    }
```

```
[root@localhost www]#service nginx reload
```

完成上述操作后，访问网站后台的添加商品页面 http://www.itshop.test/Admin/Goods/add.html，在用于编写商品详情的在线编辑器中上传图片进行测试，如图 8-14 所示。如果图片可以上传成功且能够正确显示，则说明上述操作配置成功。

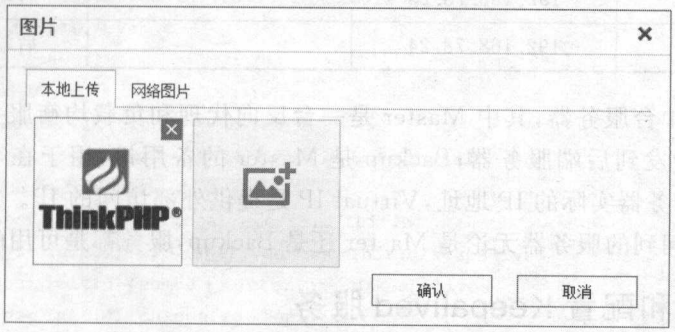


图 8-14　上传图片测试

8.3　Nginx＋Keepalived 高可用方案

8.3.1　高可用方案概述

高可用(High Availability)是指通过专门的架构设计，减少整个系统的故障时间，保持其提供服务的高度可用性。在将 Nginx 作为负载均衡服务器使用时，upstream 机制能够检测出后端服务器是否可用，如果其中一台服务器宕机，Nginx 能够自动转移到后端正常的服务器中，以保持系统持续可用。

衡量一个集群的高可用性在于没有单点故障，即其中任何一台服务器宕机都不会造成整个服务中断。若一个集群在前端只有一台 Nginx 反向代理负载均衡服务器，一旦该服务器发生故障，就会造成整个集群的服务中断。为了解决这个问题，可以利用 Keepalived 部署备用服务器，实现故障转移。

Keepalived 内置了 VRRP(Virtual Router Redundancy Protocol，虚拟路由冗余协议)功能，VRRP 用于解决静态路由出现的单点故障问题，它通过 IP 多播的方式通信，当发现主路由故障时，通过选举策略将备用路由更换为主路由，从而继续提供服务。

Keepalived 利用 VRRP 实现了将提供对外访问的 IP 地址(Virtual IP)自动在主服务器(Master)和备用服务器(Backup)之间切换，正常情况下 Master 使用 Virtual IP 提供对外访问，当 Master 故障时，其他正在监控 Master 的 Backup 会通过优先级(priority)机制竞争接管 Virtual IP 继续对外提供服务，其他落选的 Backup 会继续监控当前使用的 Virtual IP 服务器。

下面利用 VMware 搭建基于 Nginx＋Keepalived 的高可用实验环境，参见表 8-4。

表 8-4 高可用环境中的服务器

角 色	RIP(Real IP)	VIP(Virtual IP)	说 明
Master	192.168.78.21	192.168.78.20	Nginx+Keepalived
Backup	192.168.78.22	192.168.78.20	Nginx+Keepalived
—	192.168.78.23	—	后端服务器 1
—	192.168.78.24	—	后端服务器 2

表 8-4 中有 4 台服务器，其中 Master 是一台反向代理和负载均衡服务器，用于对外提供访问，将请求转发到后端服务器；Backup 是 Master 的备用机，用于在 Master 故障时代替。Real IP 是服务器实际的 IP 地址，Virtual IP 是提供外部访问的 IP。对客户端来说，通过 Virtual IP 访问到的服务器无论是 Master 还是 Backup，服务都是可用的。

8.3.2 安装和配置 Keepalived 服务

1. 部署主服务器

请读者自行创建一个虚拟机，采用最小化方式安装 CentOS 6.8，设置 IP 地址为 192.168.78.21。然后编译安装 Nginx，更改防火墙规则允许 80 端口外部访问。

然后在 Nginx 的站点目录下编写一个 index.html 文件，用于提示当前服务器身份，具体步骤如下。

```
#① 创建 index.html 文件
[root@localhost ~]#cd /usr/local/nginx/html
[root@localhost html]#echo 'This is Master' >index.html
#② 访问测试
[root@localhost html]#curl http://localhost
This is Master
```

2. 获取和安装 Keepalived

在 Keepalived 软件的官方网站 http://www.keepalived.org 可以获取下载地址，如图 8-15 所示。

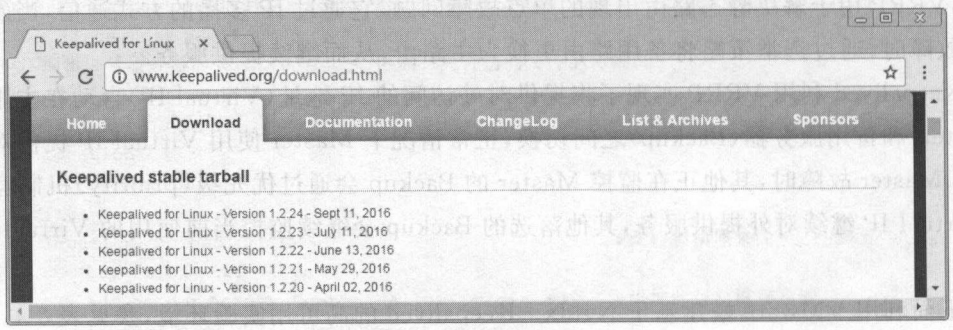


图 8-15 获取 Keepalived

将 Keepalived 源码包下载到服务器中，然后按照如下操作步骤进行安装即可。

```
#① 解压文件
[root@localhost ~]#tar -zxvf keepalived-1.2.24.tar.gz
[root@localhost ~]#cd keepalived-1.2.24
#② 编译安装
[root@localhost keepalived-1.2.24]#./configure
    #检查确保如下信息为 yes
    Use VRRP Framework            : Yes
    Use VRRP VMAC                 : Yes
    Use VRRP authentication       : Yes
[root@localhost keepalived-1.2.24]#make && make install && cd ..
#③ 添加到系统服务，配置开机启动
[root@localhost ~]#cd /usr/local/etc/rc.d/init.d
[root@localhost init.d]#cp keepalived /etc/init.d/keepalived
[root@localhost init.d]#chmod +x /etc/init.d/keepalived
[root@localhost init.d]#chkconfig keepalived on
#④ 链接配置文件
[root@localhost init.d]#cd /usr/local/etc/sysconfig
[root@localhost sysconfig]#ln -s `pwd`/keepalived /etc/sysconfig/keepalived
#⑤ 链接程序文件
[root@localhost sysconfig]#cd /usr/local/sbin
[root@localhost sbin]#ln -s `pwd`/keepalived /usr/sbin/keepalived
```

3. 配置主服务器的 Keepalived

启动 Keepalived 前需要创建一个配置文件，Keepalived 将会根据配置文件中的配置进行工作。执行如下命令可以查看 Keepalived 的配置文件模板，该文件中提供了一些参考配置。

```
[root@localhost ~]#less /usr/local/etc/keepalived/keepalived.conf
```

在 Keepalived 服务脚本中，默认加载的配置文件路径为/etc/keepalived/keepalived.conf，目前该文件并不存在，需要手动创建，具体操作步骤如下。

```
[root@localhost ~]#mkdir /etc/keepalived
[root@localhost ~]#vi /etc/keepalived/keepalived.conf
```

接下来编写配置文件，为主服务器(Master)配置 Keepalived 服务，编写内容如下。

```
vrrp_instance VI_1 {                    #配置一个虚拟路由，名称为 VI_1
    state MASTER                        #指定 Keepalived 的角色，MASTER 或 BACKUP
    interface eth0                      #指定监测的网卡
    virtual_router_id 21                #虚拟路由的标识，同一个 VRRP 的 MASTER 和 BACKUP 应一致
    mcast_src_ip 192.168.78.21          #设置 Real IP(可省略，默认将自动使用网卡的主 IP)
    priority 100                        #优先级、权重(权重最高的主机将接管 Virtual IP)范围 0~254
    advert_int 1                        #MASTER 和 BACKUP 之间同步检查的时间间隔，单位秒
    authentication {                    #设置验证类型和密码
        auth_type PASS                  #验证类型，PASS 表示使用密码验证
```

```
        auth_pass 123456                #设置密码,用于 MASTER 和 BACKUP 之间使用相同密码通信
    }
    virtual_ipaddress {                 #设置 Virtual IP 地址池,每行一个
        192.168.78.20                   #为 MASTER 和 BACKUP 设置相同的 Virtual IP
    }
}
```

完成上述配置文件的编写后,即可启动 Keepalived 服务,并检查配置是否生效。具体步骤如下。

```
#① 启动 Keepalived 服务
[root@localhost ~]# service keepalived start
Starting keepalived:       [  OK  ]
#② 查看 Keepalived 进程
[root@localhost ~]#ps aux | grep keepalived
root      5467  0.0  0.4  35268   980 ?        Ss   04:45   0:00 keepalived -D
root      5468  0.1  0.8  37372  2128 ?        S    04:45   0:00 keepalived -D
root      5470  1.9  0.6  37372  1552 ?        S    04:45   0:00 keepalived -D
#③ 查看 Keepalived 添加的 Virtual IP
[root@localhost sbin]#ip a | grep 192.168.78.20
    inet 192.168.78.20/32 scope global eth0
```

在 Keepalived 启动后,就可以使用 IP 地址 192.168.78.20 访问本服务器。若使用物理机浏览器进行访问,可以看到之前编写的 index.html 的网页内容,如图 8-16 所示。

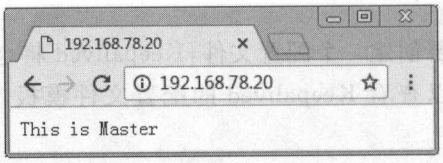


图 8-16 测试 Virtual IP

4. 配置备用服务器的 Keepalived

基于主服务器克隆出一台备用服务器,将 IP 地址配置为 192.168.78.22。然后打开 Keepalived 配置文件,将服务器身份更改为 BACKUP,并降低优先级,具体修改如下。

```
vrrp_instance VI_1 {
    state BACKUP                #修改身份为 BACKUP
    priority 90                 #修改优先级为 90(低于 MASTER 即可)
    ⋮
}
```

Master 和 Backup 服务器中的 Keepalived 通过 VRRP 的 112 端口通信,若端口无法访问则会同时抢占 Virtual IP 地址。接下来为两台服务器配置防火墙规则,开放 112 端口,具体操作如下。

```
#① 在 Backup 服务器中允许 Master 访问 112 端口
[root@localhost ~]#iptables -I INPUT -s192.168.78.21 -p112 -jACCEPT
[root@localhost ~]#service iptables save
#② 在 Master 服务器中允许 Backup 访问 112 端口
[root@localhost ~]#iptables -I INPUT -s192.168.78.22 -p112 -jACCEPT
[root@localhost ~]#service iptables save
```

完成上述配置后，执行 service keepalived start 启动 Backup 服务器的 Keepalived 服务。为了区分当前访问到的是哪一台服务器，在备用服务器中修改 index.html 文件。

```
[root@localhost ~]#cd /usr/local/nginx/html
[root@localhost html]#echo 'This is Backup' >index.html
```

通过浏览器访问 http://192.168.78.20 进行测试。由于 Master 的优先级(priority 100)高于 Backup(priority 90)，因此访问到的一直是 Master 服务器。当在 Master 中执行 service network stop 命令停止网络时，再次访问就会看到网页内容变为 This is Backup，说明当前 Backup 已经接管成功。

在 Master 中执行 service network start 命令恢复网络后，192.168.78.20 的访问结果又会变为 This is Master，这是因为 Master 能够抢占 Backup 接管的 Virtual IP。

另外，Keepalived 还可以配置多台 Backup 服务器，从而进一步提高服务的可用性。

8.3.3　使用 Keepalived 监控 Nginx 服务

使用 Keepalived 除了可以监控其他服务器中的 Keepalived 是否正常，也可以监控本机中的某个服务是否正常。下面将实现利用 Keepalived 监控 Nginx 进程是否正常工作，如果 Nginx 出现问题，则尝试重新启动 Nginx。如果重启 Nginx 后仍然无法工作，则停止本机的 Keeplived 服务，从而使其他备用服务器自动接管 Virtual IP。

1. 自动监控 Nginx

在 Master 和 Backup 服务器中修改 Keepalived 配置文件，编写内容如下。

```
vrrp_script chk_nginx {                    #配置用于检测 Nginx 运行状态的脚本
    script "/chk_nginx.sh"                 #用于检测的脚本文件路径
    interval 2                             #每 2 秒执行一次脚本
    weight -20                             #当检测失败时，权重发生的变化
}
vrrp_instance VI_1 {                       #为 VI_1 添加监控脚本
    ⋮
    track_script {
        chk_nginx
    }
}
```

接下来执行 vi /chk_nginx.sh 命令创建监控脚本，编写代码如下。

```
1    #!/bin/bash
2    if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
```

```
3        service nginx start
4        sleep 2
5        if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
6            service keepalived stop
7        fi
8    fi
```

上述代码中第2行判断当前Nginx进程数量是否为0，为0表示Nginx没有启动，则执行第3行的命令启动Nginx，然后等待2秒后再次检测Nginx是否启动，如果仍然没有启动则停止Keepalived服务。

完成上述脚本编写后，为脚本设置可执行权限，然后重新加载配置。

```
[root@localhost ~]#chmod +x /chk_nginx.sh
[root@localhost ~]#service keepalived reload
```

2. Nginx+Keepalived高可用测试

为了测试Keepalived是否能够实现Nginx服务的高可用，下面通过手动停止Nginx服务的方式进行验证。按照如下步骤进行操作即可。

```
#① 停止Nginx服务
[root@localhost ~]#service nginx stop
#② 等待2秒后，查看Nginx是否已经恢复启动
[root@localhost ~]#ps -C nginx --no-header
  3617 ?        00:00:00 nginx
  3618 ?        00:00:00 nginx
```

从上述执行结果中可以看出，当前配置的Keepalived能够检测出Nginx已经停止，并自动恢复。

接下来继续测试当Nginx无法重新启动时，Keepalived能否将Virtual IP自动切换到备用服务器。首先保证在正常情况下浏览器访问http://192.168.78.20看到的结果是This is Master，然后在主服务器中执行如下命令停止Nginx服务并立即取消Nginx程序的可执行权限，具体操作如下。

```
#① 创建停止Nginx服务并立即取消Nginx程序执行权限的脚本
[root@localhost ~]#cd /usr/local/nginx/sbin
[root@localhost sbin]#vi test.sh
    #编写内容如下：
    #!/bin/bash
    service nginx stop
    chmod -x nginx
#② 执行脚本(测试完成后，使用"chmod +x nginx"恢复执行权限)
[root@localhost sbin]#chmod +x test.sh
[root@localhost sbin]#./test.sh
```

执行上述命令后，通过浏览器访问进行测试，若网页显示的结果变为This is Backup，

就说明当前已经自动切换到备用服务器。至此，Nginx＋Keepalived 高可用环境已经部署完成，当正常提供对外访问的服务器发生宕机、Nginx 无法启动等故障时，另外一台备用服务器会自动继续提供服务。

本章小结

本章首先讲解 Nginx 的一些实用的配置优化，然后搭建一个 LNMP 分布式集群，最后讲解基于 Nginx＋Keepalived 的高可用环境。通过本章的学习，读者应该能够掌握 Nginx 在集群环境下的部署、配置和应用，学会利用集群、分布式、高可用等技术来提高网站系统的负载能力、并发能力和可靠性。

课后练习

一、填空题

1. _____是指将多台服务器集中起来一起进行同一种服务。
2. 修改 Linux 系统的文件限制命令写在_____文件中，可以实现开机后自动修改。

二、判断题

1. Keepalived 配置文件中 priority 的权重值越大则其优先级越高。（　　）
2. 由于 Cookie 是客户端技术，所以它将用户的数据保存在用户浏览器的缓存目录下。（　　）
3. Nginx 服务器默认情况下允许的并发连接数为 1000 个。（　　）
4. 当服务器需要更新静态资源时，可以通过刷新网页重新请求静态资源。（　　）
5. 默认情况下 NFS 文件服务器安装后，自动创建的用户和用户组是 nobody。（　　）

三、简答题

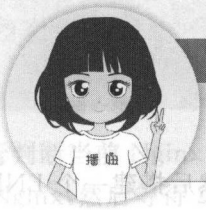
1. 请简述集群与分布式的异同点。
2. 阅读下面的 Memcached 服务的配置文件，分析各选项的含义。

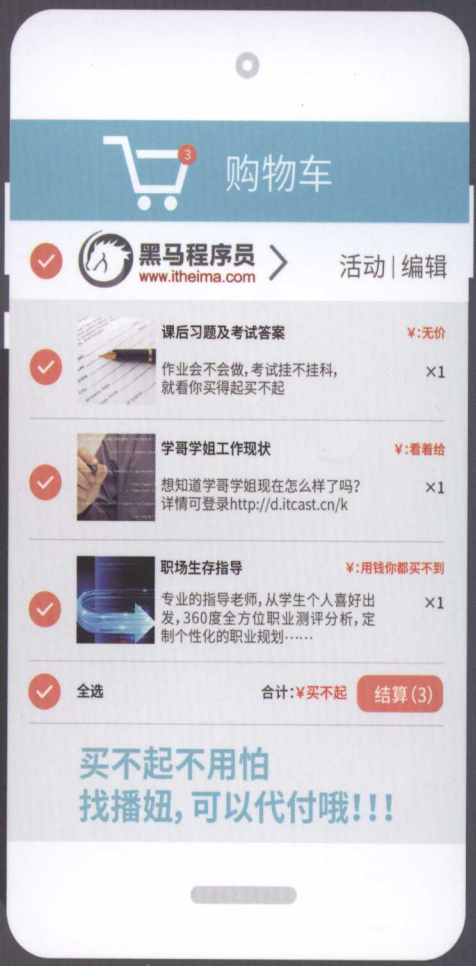
```
-m 512
-p 11211
-u nobody
-c 1024
-t 16
```

四、操作题

在部署集群时，每台后端 Web 服务器的站点文件都是相同的，在发布和更新文件时会遇到大量的重复操作。为此，可以利用 rsync＋inotify 实现文件实时同步，当一台服务器的

文件内容发生变化时，其他服务器就会自动进行同步。请尝试实现此功能。





国家信息技术紧缺人才培养工程指定教材

移动互联网开发类

《Android移动应用基础教程》	978-7-113-19620-2
《Android项目实战——手机安全卫士》	978-7-113-20549-2
《Objective-C入门教程》	978-7-115-35625-3
《iOS开发项目化入门教程》	978-7-115-29949-9
《iOS开发项目化经典教程》	978-7-115-41074-0

互联网程序设计类

《Java基础入门》	978-7-302-35938-8
《Java Web程序开发入门》	978-7-302-38794-7
《Java Web程序开发进阶》	978-7-302-40726-3
《MySQL数据库入门》	978-7-302-38795-4
《SSH框架整合实战教程》	978-7-302-42389-8
《C语言开发入门教程》	978-7-115-35623-9
《C语言程序设计教程》	978-7-113-19570-0
《C++程序设计教程》	978-7-115-39484-2
《C#程序设计基础入门教程》	978-7-115-35624-6
《Nginx高性能Web服务器实战教程》	978-7-302-47244-5

UI前端设计类

《网页设计与制作(HTML+CSS)》	978-7-113-18580-0
《HTML+CSS+JavaScript网页制作案例教程》	978-7-115-29658-0
《HTML5+CSS3网站设计基础教程》	978-7-115-41064-1
《Photoshop CS6图像处理案例教程》	978-7-113-21208-7
《PHP程序设计基础教程》	978-7-113-18570-1
《PHP程序设计高级教程》	978-7-113-19571-7
《PHP网站开发实例教程》	978-7-115-29576-7
《PHP+Ajax+jQuery网站开发项目式教程》	978-7-115-41075-7

扫一扫，码上查看

扫一扫

课件下载、样书申请
教材推荐、技术交流

ISBN 978-7-302-47244-5

9 787302 472445 >

定价：45.00元